Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка веб-сервісу взаємодії з блокчейном Ethereum для фінансової платформи мовою Python

Виконав(ла): студент(ка) 6 курсу, групи СПм-62
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Орлов В.С. (прізвище та ініціали)

Керівник (підпис) Ясній О.П. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)

« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Орлову Володимиру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-сервісу взаємодії з блокчейном Ethereum для фінансової платформи мовою Python

Керівник роботи Ясній Олег Петрович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи вимоги до функціоналу системи, веб-сервіс, взаємодія з блокчейном Ethereum

4. Зміст роботи (перелік питань, які потрібно розробити)
Спроектувати та розробити систему для оброблення транзакцій у блокчейні Ethereum. Система повинна вміти автоматично зараховувати депозити, відправляти транзакції та регулювати гаманцями у системі. Дослідити і розробити можливість інтеграції зі сторонніми платформами.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма взаємодії підсистем, діаграма варіантів використання, діаграма послідовностей, діаграма послідовностей, слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Галина Михайлівна		
Безпека в надзвичайних ситуаціях	Стручок Володимир Сергійович		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	19.09.2023	
2	Огляд існуючих методів	26.09.2023	
3	Основна частина	15.10.2023	
4	Підготовка пояснювальної записки	25.11.2023	
5	Спецчастина	28.11.2023	
6	Підготовка презентації та доповіді	03.12.2023	
7	Попередній захист	13.12.2023	
8	Нормоконтроль, рецензування		
9	Занесення диплома в електронний архів		
10	Допуск до захисту у зав. кафедри		

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Проект на тему «Розробка веб-сервісу взаємодії з блокчейном Ethereum для фінансової платформи мовою Python».

Орлов Володимир Сергійович. Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–62, Тернопіль – 2023.С. – 106, рисунків – 26, додатків – 4.

Мета проекту: розробити веб-сервіс для взаємодії з блокчейном Ethereum, який дозволяє управляти гаманцями, обробляти транзакції та моніторити стан блокчейну. Для створення цього веб-сервісу було використано Python та FastAPI. Як систему керування базами даних обрано PostgreSQL, а також використано Docker для контейнеризації сервісу. Користь проекту полягає у зростаючій потребі сучасного ринку у безпечних і ефективних рішеннях для обігу цифрових активів. Веб-сервіс значно спрощує процеси взаємодії з блокчейном, економить час користувачів та підвищує безпеку їхніх операцій. Розроблена система дозволяє швидко та ефективно виконувати транзакції в Ethereum, забезпечуючи зручний інтерфейс для користувачів. Особливу увагу приділено оптимізації швидкості обробки запитів і забезпеченню високого рівня безпеки даних користувачів.

ABSTRACT

Project on "Development of a web service for interaction with the Ethereum blockchain for a financial platform in Python".

Orlov Volodymyr Serghiyovych. Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SPm-62, Ternopil - 2023.C. – 106, drawings – 26, applications – 4.

Project goal: to develop a web service for interacting with the Ethereum blockchain, which allows you to manage wallets, process transactions and monitor the state of the blockchain. Python and FastAPI were used to create this web service. PostgreSQL was chosen as the database management system, and Docker was used to containerize the service. The project's benefit lies in the growing need of the modern market for secure and efficient solutions for the circulation of digital assets. The web service greatly simplifies the processes of interacting with the blockchain, saves users' time and increases the security of their operations. The developed system allows for fast and efficient execution of transactions in Ethereum, providing a user-friendly interface. Particular attention is paid to optimizing the speed of request processing and ensuring a high level of user data security.

ЗМІСТ

ВСТУП

У сучасному світі, де технології стрімко розвиваються та змінюють звичні парадигми ведення бізнесу, впровадження інноваційних рішень стає не просто перевагою, а необхідністю для успішної конкуренції та розвитку. Одним з найбільш проривних трендів цифрової ери є блокчейн-технології, які набули широкої популярності з появою біткоїна. Ці технології відкривають нові горизонти не лише у фінансовому секторі, а й у різних аспектах економічного та соціального життя.

Блокчейн Ethereum з його функціоналом смарт-контрактів і децентралізованими додатками виходить за рамки традиційного розуміння криптовалют, пропонуючи рішення для різноманітних практичних застосувань – від фінансових транзакцій до управління цифровою ідентифікацією та логістики. З цієї причини інтеграція блокчейну в бізнес-процеси та створення веб-сервісів, що дозволяють взаємодіяти з цією технологією, стає ключовим завданням для сучасних компаній.

Впровадження блокчейн-технологій у сучасний бізнес-простір відкриває нові перспективи для розвитку, забезпечуючи більшу прозорість, безпеку та ефективність у різних галузях. Особливої уваги заслуговують смарт-контракти Ethereum, які дозволяють автоматизувати процеси, мінімізувати ризики та знизити транзакційні витрати. Ці особливості роблять Ethereum ідеальним для використання в бізнес-додатках, які вимагають високого ступеня надійності та автономності. Таким чином, інтеграція Ethereum у веб-сервіси відкриває широкі можливості для інновацій, створюючи нові шляхи для розвитку бізнесу в цифрову епоху.

Крім технічних аспектів, розробка веб-сервісу для взаємодії з блокчейном Ethereum важлива і з соціально-економічної точки зору. З огляду на глобалізацію економіки та зростання цифрового впливу, потреба в безпечних і надійних цифрових платформах є більш важливою, ніж будь-коли. Використання

блокчейну може сприяти підвищенню довіри між учасниками ринку, зменшенню бюрократичних бар'єрів та відкриттю нових можливостей для малого та середнього бізнесу, який прагне вийти на міжнародний рівень. Таким чином, розробка веб-сервісу для взаємодії з Ethereum не лише відповідає на технічні виклики, а й сприяє соціально-економічному розвитку, підтримуючи інноваційний потенціал та конкурентоспроможність бізнесу.

Історично блокчейн виник як відповідь на необхідність створення децентралізованої, безпечної та незалежної від єдиного регулятора системи ведення рахунків. З часом він перетворився на універсальний інструмент, який пропонує нові можливості для інновацій у багатьох сферах. У цьому контексті розробка веб-сервісу, що використовує потенціал Ethereum, є не лише технічним викликом, але й важливим кроком на шляху інтеграції цифрових технологій у сучасний бізнес.

Дана дипломна робота присвячена розробці веб-сервісу для взаємодії з блокчейном Ethereum, що відповідає сучасним вимогам цифрової економіки. Основною метою є створення надійного, ефективного та зручного бізнес-інструменту, що дозволяє інтегрувати блокчейн-технології в повсякденну діяльність. Важливість та актуальність цієї теми обумовлена широким спектром застосування блокчейну у фінансовому секторі, управлінні активами, прозорості та безпеці даних.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Знайомство зі світом блокчейну починається з розуміння його основ. Блокчейн – це децентралізована технологія, яка змінює спосіб зберігання та обробки інформації, пропонуючи новий рівень безпеки та прозорості. Ця технологія використовує послідовно з'єднані блоки даних, які забезпечують незмінність інформації та надійність системи.

Ефіріум, як один з провідних представників блокчейн-технологій, відрізняється використанням смарт-контрактів. Ці контракти є автономними програмами, які автоматично виконують умови угод, запрограмованих в блокчейні. Ethereum пропонує не тільки нові можливості для фінансових транзакцій, а й широкий спектр застосувань, від децентралізованих додатків до систем голосування та ідентифікації.

Загальні поняття блокчейну та Ефіріуму важливі для розуміння потенціалу та обмежень цих технологій. Вони відіграють ключову роль у розвитку цифрової економіки, пропонуючи нові можливості для безпечних, прозорих та ефективних цифрових транзакцій. Зокрема, вони відкривають нові горизонти для розвитку веб-сервісів, здатних взаємодіяти з блокчейнами, роблячи їх невід'ємною частиною сучасної цифрової інфраструктури.

У цьому контексті глибоке розуміння теоретичних основ блокчейну та Ethereum є необхідним для розробки ефективних, надійних та інноваційних веб-сервісів, здатних реалізувати потенціал цих технологій у реальних бізнес-процесах. Вивчення цих основ допомагає зрозуміти, як максимально використати потенціал блокчейну для вирішення конкретних завдань, що стоять перед сучасними компаніями, а також дозволяє оцінити виклики та обмеження, які супроводжують впровадження цих технологій.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Теоретичні основи блокчейну

Блокчейн – це розподілена база даних, яка зберігає інформацію у вигляді ланцюжка блоків. Кожен блок містить інформацію про попередній блок, а також дані, які необхідно зберегти. Дані в блокчейні захищені криптографією, що робить їх дуже складними для зміни або фальсифікації. Одним з ключових аспектів блокчейну є його децентралізація. Це означає, що немає центрального сервера, який керує блокчейном. Замість цього блокчейн підтримується мережею комп'ютерів, які називаються вузлами.

Децентралізація дає ряд переваг. По-перше, вона робить блокчейн більш надійним і безпечним. Якщо один вузол буде зруйновано або викрадено, це не вплине на цілісність блокчейну. По-друге, децентралізація підвищує прозорість і підзвітність. Будь-хто може перевірити інформацію в блокчейні без необхідності довіряти жодному центральному органу.

Ще однією важливою характеристикою блокчейну є його незмінність. Це означає, що дані в блокчейні не можуть бути змінені без порушення цілісності інших блоків. Незмінність забезпечується використанням хеш-функцій.

Хеш-функція – це математична функція, яка перетворює дані в унікальний і незмінний код, яка обчислюється для кожного блоку в ланцюжку. Результат включається у наступний блок, створюючи залежну систему. У випадку блокчейну вона обчислюється на основі наступних даних:

1. Попередній блок – ідентифікатор попереднього блоку в ланцюжку.
2. Номер блоку – унікальний ідентифікатор блоку.
3. Мітка часу – дата і час створення блоку.
4. Дані – інформація, що зберігається в блоці.

Щоб змінити інформацію в блокчейні, потрібно змінити хеш-код кожного блоку, починаючи з блоку, який містить змінену інформацію. Зробити це дуже

складно, оскільки хеш-код блоку обчислюється на основі всіх даних, що зберігаються в блоці. Ось як працює процес хешування блоків:

1. Хеш-функція приймає дані блоку на вхід.
2. Хеш-функція обчислює вихідний код, який називається хеш-кодом.
3. Хеш-код блоку включається в наступний блок.

Важливою характеристикою хеш-функцій є те, що вони є односторонніми. Це означає, що хеш-код блоку можна легко обчислити, але, навпаки, дані блоку не можуть бути відновлені на основі його хеш-коду.

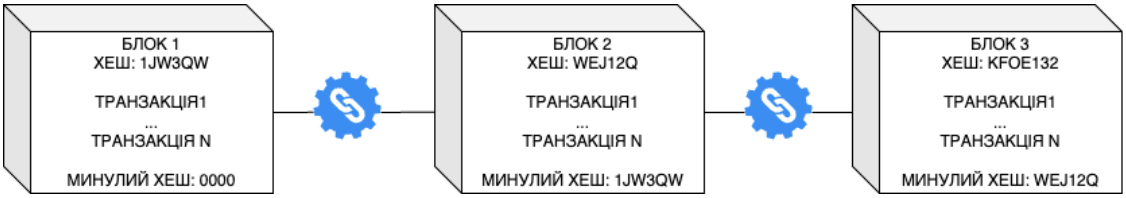


Рисунок 1.1 – Демонстрація пов’язаності блоків у блокчейні

Блокчейн має широкий спектр потенційних застосувань. Його можна використовувати для:

1. Зберігання фінансових даних, таких як криптовалютні транзакції, рахунки та контракти.
2. Управління ланцюгами поставок для забезпечення прозорості та підзвітності.
3. Забезпечення прозорості та підзвітності в інших сферах, таких як енергетика, охорона здоров'я та демократія.
4. Створення нових форм співпраці між організаціями.

Це відносно нова технологія, яка все ще перебуває в процесі розвитку. Проте вона має значний потенціал для зміни багатьох сфер нашого життя. Одна з головних тенденцій розвитку блокчейну – його масштабованість. Для того, щоб блокчейн можна було використовувати для масштабних додатків, необхідно розробляти більш ефективні алгоритми шифрування та зберігання даних. Ще одна важлива тенденція – розробка нових типів блокчейну. Сьогодні існують різні типи

блокчейнів, які призначені для різних цілей. У майбутньому, ймовірно, будуть розроблені нові типи блокчейнів, які матимуть ще ширший спектр застосування.

1.2.2 Різновиди та особливості блокчейнів

Існує кілька різних типів блокчейнів [4], які відрізняються за своїми характеристиками та призначенням. Найпоширеніші типи блокчейнів [11]:

- Публічний блокчейн – відкритий для всіх. Будь-хто може приєднатися до публічної мережі блокчейн і генерувати нові блоки. Публічні блокчейни часто використовуються для зберігання фінансових даних, наприклад, криптовалютних транзакцій.
- Приватний блокчейн – доступний лише авторизованим користувачам. Приватні блокчейни часто використовуються для корпоративних додатків, таких як управління ланцюгами поставок або управління правами власності.
- Гібридний блокчейн – поєднує в собі характеристики публічного та приватного блокчейнів. Гібридний блокчейн часто використовується для додатків, які вимагають одночасно відкритості та безпеки.

Також важливо відмітити низку характерних особливостей, які відрізняють блокчейн від інших технологій зберігання даних. До таких особливостей відносяться:

- Децентралізація – це розподіл влади або контролю між кількома суб'єктами. У контексті блокчейну децентралізація означає, що немає центрального сервера, який керує мережею. Замість цього мережа тримається на мережі вузлів, які розподілені по всьому світу. Вона забезпечує надійність, безпеку, прозорість та стійкість даних.
- Незмінність – це властивість блокчейну, яка означає, що дані в блокчейні не можуть бути змінені без шкоди для цілісності інших блоків. Незмінність забезпечується використанням хеш-функцій.

- Достовірність – це якість блокчейну, яка випливає з двох попередніх властивостей (децентралізація та незмінність) та забезпечує правдивість і точність інформації.

1.2.3 Алгоритми консенсусу

Алгоритми консенсусу – це механізми, які використовуються в блокчейні для досягнення згоди між учасниками мережі щодо стану блокчейну. Вони визначають, як нові блоки додаються до ланцюжка і хто отримує винагороду за їх створення. Існує багато різних алгоритмів консенсусу, що використовуються в блокчейні.

Незважаючи на переваги цього механізму, деякі алгоритми консенсусу можуть бути енергоємними або повільними. Наприклад, алгоритм Proof-of-Work, що використовується в Bitcoin, вимагає від учасників мережі розв'язання складних математичних задач, що може призвести до значного споживання енергії. Інші алгоритми консенсусу, такі як Proof-of-Stake, є більш енергоефективними, але вони можуть працювати повільніше. Вибір алгоритму консенсусу залежить від конкретних потреб мережі. Наприклад, мережа, яка вимагає високої безпеки, може вибрати алгоритм Proof-of-Work, навіть якщо він енергоємний. Мережа, яка потребує більшої енергоефективності, може вибрати алгоритм Proof-of-Stake.

Список найпоширеніших алгоритмів з консенсусу наведено з подробицями нижче:

- Proof-of-Work (PoW) – це алгоритм, який використовує складні математичні розрахунки для перевірки транзакцій і створення нових блоків. Для того, щоб додати новий блок до ланцюжка, учасники мережі повинні вирішити складну математичну задачу. Перший учасник, який вирішив

задачу, отримує винагороду у вигляді нових монет. Використовується в таких блокчейнах як: Bitcoin, Litecoin, Dogecoin, Monero.

- Proof-of-Stake (PoS) – це алгоритм, який використовує частку активів учасника мережі для перевірки транзакцій і створення нових блоків. Учасники мережі з більшою кількістю активів мають більше шансів бути обраними для створення нового блоку. Використовується в таких блокчейнах як: Cardano, Solana, Ethereum.
- Delegated Proof-of-Stake – модифікація алгоритму PoS, в якій учасники мережі обирають делегатів, відповідальних за створення нових блоків. Використовується в таких блокчейнах як: TRON, EOS.
- Proof-of-Authority (PoA) – алгоритм, в якому створення нових блоків доручається обмеженій кількості авторитетних вузлів. Використовується в блокчейні EOSIO.

1.2.4 Принцип роботи блокчейну Ethereum

Блокчейн Ethereum – це децентралізована платформа, яка дозволяє створювати децентралізовані додатки (DApps). DApps можна використовувати для різних цілей, включаючи фінансові послуги, ігри та смарт-контракти. Він використовує алгоритм консенсусу Proof-of-Stake (PoS). Згідно з цим алгоритмом, учасники мережі, які мають більше активів Ethereum (ETH), мають більше шансів бути обраними для створення нових блоків. Нові блоки додаються до блокчейну Ethereum в середньому кожні 15 секунд. Кожен блок містить інформацію про всі транзакції, які відбулися за цей період [12].

Щоб здійснити транзакцію в мережі Ethereum, користувач повинен створити запит на транзакцію. Запит на транзакцію містить інформацію про відправника, одержувача та суму коштів. Запит на транзакцію відправляється в мережу Ethereum. Він розподіляється по мережі і збирається в блок. Коли блок створено,

він шифрується за допомогою криптографії. Це гарантує, що транзакції в блоці не можуть бути змінені або видалені. Блок додається до блокчейну Ethereum і стає загальнодоступним. Згідно з принципом децентралізації, всі учасники мережі Ethereum мають доступ до блокчейну. Це означає, що будь-який користувач може перевірити транзакції, які відбулися в мережі.

Також не менш важливою особливістю блокчейну Ethereum є його логіка смарт-контрактів. Смарт-контракти – це програми, які зберігаються в блокчейні Ethereum і можуть виконуватися без втручання людини. Їх можна використовувати для різних цілей, зокрема:

- Фінансові послуги: смарт-контракти можна використовувати для створення децентралізованих банків, бірж та інших фінансових інструментів, для прикладу фінансова платформа віртуальних активів.
- Ігри: смарт-контракти можна використовувати для створення децентралізованих ігор, в яких гравці можуть володіти своїми активами, для прикладу маркет продажу віртуальних предметів
- Управління: смарт-контракти можна використовувати для створення децентралізованих систем управління, які не потребують довіри до центральної влади, для прикладу служба доставки посилок.

Смарт-контракти в Ethereum написані на мові програмування Solidity. Solidity – це об'єктно-орієнтована мова програмування, яка спеціально розроблена для смарт-контрактів. Вони є безпечними, оскільки вони зберігаються у блокчейні та мають свою унікальну адресу. Також вони є децентралізованими, оскільки вони не потребують довіри до центрального органу. Смарт-контракти можуть виконуватися без втручання людини, що робить їх більш ефективними і безпечними.

1.2.5 Принцип роботи транзакцій у блокчейні Ethereum

Транзакції в блокчейні Ethereum – це зміни в стані мережі, які здійснюють користувачі. Вони можуть бути використані для таких цілей, як переказ коштів, створення смарт-контрактів або виконання інших операцій. Транзакція складається з наступних елементів:

- Тип – визначає тип транзакції. Наприклад, транзакції грошових переказів мають тип "переказ".
- Джерело – адреса відправника транзакції.
- Призначення – адреса одержувача транзакції.
- Дані – додаткові дані, які можуть бути пов'язані з транзакцією. Наприклад, при переказі коштів даними можуть бути сума переказу та метадані.
- Підпис – цифровий підпис відправника транзакції. Підпис використовується для підтвердження того, що відправник уповноважений здійснювати транзакцію.

Коли користувач створює транзакцію, вона відправляється в мережу Ethereum. Транзакції зберігаються в мемпулі – тимчасовому сховищі транзакцій, які ще не були включені в блокчейн. Майнери збирають транзакції з мемпула і створюють блоки. Блоки містять набори транзакцій, які були підтверджені майнерами. Блоки додаються до блокчейну, який є хронологічним списком блоків, що зберігають всі транзакції, які були проведені в мережі Ethereum.

За кожну транзакцію в Ethereum стягується комісія. Комісія – це платіж, який виплачується власникам за включення транзакції в блок. Розмір комісії залежить від таких факторів, як розмір транзакції, складність блоку і поточний рівень використання мережі. Комісія за транзакцію Ethereum є важливим елементом, який допомагає підтримувати безпеку мережі. Вона мотивує майнерів включати транзакції в блоки в першу чергу. Це допомагає запобігти атакам, таким як переповнення мережі транзакціями.

Транзакції Ethereum можна скасувати, але це складний процес. Щоб відмінити транзакцію, відправник повинен створити нову транзакцію, яка

скасовує першу. Ця нова транзакція повинна мати вищу комісію, ніж перша. Якщо нова транзакція буде підтверджена майнерами раніше, ніж перша, вона скасує першу.

Ви також можете прискорити транзакцію, збільшивши комісію. Це збільшить ймовірність того, що транзакція буде включена в блок раніше. Однак це також збільшить вартість транзакції.

1.3 Постановка задачі

Дослідження, покликані зробити внесок у науку шляхом систематичного аналізу, інтерпретації та оцінки даних, зазвичай називають науковими дослідженнями. У таких дослідженнях основна увага зазвичай зосереджена на процесі збору знань, а не на матеріальній вигоді. Результати, отримані невеликою командою під час таких досліджень, публікуються, що призводить до відкриття нової інформації в галузі діагностики, лікування та надійності використання.

У світі, що стрімко розвивається, де цифрові технології змінюють обличчя традиційних галузей, впровадження інноваційних рішень стає ключовою перевагою для бізнесу в усіх секторах економіки. Історія технології блокчейн, яка починається з винайдення біткоїна у 2008 році, відкрила нову еру у сфері цифрової безпеки та фінансових транзакцій. Ефіріум, як розвинена форма блокчейну, що з'явилася пізніше, пропонує ще ширші можливості завдяки використанню смарт-контрактів і децентралізованих додатків.

Основна мета цього огляду – надати інформацію про визначення, класифікацію та методологію наукових досліджень. Перед початком такого дослідження дослідник повинен чітко визначити об'єкт, розробити план та обрати методологію. У Гельсінській декларації зазначено, що медичні дослідження за участю людей-добровольців мають на меті зрозуміти причини, розвиток і наслідки захворювань, а також розробити захисні, діагностичні та терапевтичні

заходи. Навіть найефективніші методи повинні постійно оцінюватися через дослідження надійності, ефективності, результативності, доступності та якості.

У цьому контексті постановка проблеми моєї дисертації стає актуальною не тільки для технологічного прогресу, але й для відповіді на потреби сучасного бізнесу, який прагне скористатися перевагами блокчейну для підвищення ефективності та безпеки своїх фінансових транзакцій. Метою даного проекту є розробка веб-сервісу, який дозволяє бізнесу інтегрувати платежі через блокчейн Ethereum. Основними завданнями проекту є:

1. Розробка механізму генерації гаманців: З огляду на зростаючу потребу в безпеці цифрових активів, механізм HDWallet дозволить більш ефективно управляти головними і депозитними адресами, що є фундаментальним для забезпечення безпеки і конфіденційності користувачів.
2. Система обробки транзакцій: впровадження системи обробки транзакцій забезпечить прозорість та ефективність обігу капіталу, відповідаючи на виклики, з якими стикаються сучасні платіжні системи.
3. Звітність: Розвиток інструментів детальної звітності про транзакції відіграє важливу роль у підвищенні прозорості та контролю за фінансовими потоками, що є основою для довіри та відповідальності в бізнесі.
4. Автоматизована система грошового клірингу: ця функція забезпечить оптимізацію фінансових операцій, допомагаючи підтримувати ліквідність та фінансову стабільність.

Використання сучасних технологій, таких як FastAPI [8], Python [3] та Docker [14], важливе не тільки для досягнення технічних цілей, але й для забезпечення масштабованості та гнучкості розгортання, що відкриває шлях для майбутніх інновацій у цій сфері. Важливий акцент буде зроблено на захисті даних і безпеці транзакцій, оскільки це є критично важливим в контексті технологій блокчейн. Інтеграція з цими технологіями сприятиме безперешкодному розгортанню та простоті використання розробленого веб-сервісу.

1.4 Аналіз вимог до програмного забезпечення

У цьому розділі буде проаналізовано вимоги до програмного забезпечення, описаного вище. Аналіз вимог є важливим кроком у процесі розробки програмного забезпечення. Він дозволяє зрозуміти, чого користувачі хочуть від системи, виявити можливі проблеми з вимогами і визначити, чи є вимоги достатніми для розробки системи.

Аналіз вимог до програмного забезпечення – це процес розуміння, документування та оцінки вимог до програмного забезпечення. Це важливий крок у процесі розробки програмного забезпечення, оскільки він допомагає гарантувати, що система буде відповідати потребам користувачів і бізнесу. Він покращує розуміння потреб і вимог зацікавлених сторін, таких як користувачі, бізнес-аналітики та розробники.

1.4.1 Аналіз нефункціональних вимог додатку

Нефункціональні вимоги – це вимоги, які описують, як система повинна працювати, а не що вона повинна робити. Вони визначають такі характеристики системи, як її надійність, продуктивність, безпека, сумісність, зручність використання та інші.

Для бізнесу цей процес є важливим щоб забезпечити відповідність системи потребам. Нефункціональні вимоги визначають такі характеристики системи, як її надійність, продуктивність, безпека, сумісність, зручність використання та інші. Ці характеристики є важливими для бізнесу, оскільки вони впливають на його успіх. Також це допоможе уникнути додаткових витрат і затримок. Нефункціональні вимоги повинні бути визначені та оцінені на ранній стадії процесу розробки програмного забезпечення. Це допоможе уникнути додаткових

витрат і затримок, які можуть виникнути через те, що нефункціональні вимоги визначаються або оцінюються пізніше.

Для розробників цей процес є важливим щоб розробити систему, яка відповідає вимогам бізнесу та користувачів. Аналіз нефункціональних вимог допомагає розробнику зрозуміти, що повинна робити система і як вона повинна це робити. Це допомагає розробнику спроектувати та впровадити систему, яка відповідає цим вимогам. Крім того, це допоможе уникнути додаткових витрат і затримок. Аналіз допомагає розробнику виявити потенційні проблеми на ранній стадії процесу розробки програмного забезпечення. Це може допомогти уникнути додаткових витрат і затримок, які можуть виникнути через те, що ці проблеми будуть виявлені пізніше.

Нефункціональні вимоги повинні бути описані, щоб:

- Переконатися, що всі зацікавлені сторони розуміють, що повинна робити система. Нефункціональні вимоги повинні бути описані чітко і стисло, щоб усі зацікавлені сторони могли зрозуміти, що повинна робити система.
- Переконатися, що система буде розроблена відповідно до вимог. Нефункціональні вимоги повинні бути включені в системну документацію, щоб гарантувати, що система розроблена відповідно до цих вимог.
- Сприяти тестуванню системи. Нефункціональні вимоги можуть бути використані для розробки тестових кейсів, які допоможуть переконатися, що система відповідає цим вимогам.

Для мого додатку, який взаємодіє з блокчейном Ethereum наведено перелік нефункціональних вимог:

Таблиця 1.1 – Опис нефункціональних вимог додатку

Нефункціональна вимога	Опис
Надійність	Система повинна бути доступною для користувачів 99% часу

Продовження таблиці 1.1

Нефункціональна вимога	Опис
Продуктивність	Система повинна обробляти не менше 50 запитів в секунду та не менше 10 транзакцій в секунду.
Безпека	Система повинна мати захищений алгоритм зберігання чутливих даних, такі як приватні ключі, паролі та особисті дані.
Зручність використання	Система повинна бути інтуїтивно зрозумілою та простою у використані. Також має бути доступна супутня документація до більшості частин програми та визначений алгоритм використання.
Сумісність	Система повинна бути сумісна зі стандартом REST API та зберігати зворотню сумісність.

1.4.2 Аналіз функціональних вимог додатку

Функціональні вимоги – це вимоги, які описують, що повинна робити система. Вони визначають, які функції повинна виконувати система, щоб задовольнити потреби користувачів. Вони зазвичай описуються в текстовому форматі. Вони також можуть бути представлені за допомогою діаграм, таких як діаграми варіантів використання, діаграми прецедентів, діаграми класів або діаграми послідовності.

Вимоги повинні бути конкретними, вимірюваними, досяжними, актуальними, сучасними та послідовними. Вони повинні бути достатньо детальними, щоб розробники могли зрозуміти, що повинна робити система.

Важливо описати функціональні вимоги, оскільки вони допомагають переконатися, що система розроблена так, щоб задовольняти потреби користувачів. Функціональні вимоги описують, що система повинна робити, а не як вона повинна це робити. Це допомагає гарантувати, що система розроблена відповідно до потреб користувачів. Це також зменшує ризик помилок під час процесу розробки. Добре описані функціональні вимоги допомагають розробникам зрозуміти, що повинна робити система.

Було розроблено таблицю варіантів використання для опису функціональних вимог з відповідним маркуванням кожного варіанту UC-DX, де X це номер варіанту.

Таблиця варіантів використання – це метод опису варіантів використання, який використовує таблицю для організації інформації. Вона допомагає зрозуміти вимоги до системи та описує, що система повинна робити, хто буде використовувати систему і які умови повинні бути виконані, перш ніж система зможе виконати вимогу. Таблиця варіантів використання містить такі поля:

- Ідентифікатор – унікальний ідентифікатор варіанту використання.
- Назва – короткий опис варіанту використання.
- Актор – роль, яка бере участь у варіанті використання.
- Передумови: Умови, які повинні бути виконані, перш ніж варіант використання може бути виконаний.
- Робочий процес: Послідовність дій, які виконуються для виконання варіанту використання.
- Пост-умови – умови, які існують після виконання варіанту використання.

Таблиця 1.2 – Варіант використання UC-D1

Атрибут	Значення
Ідентифікатор	UC-D1
Назва	Створення користувача

Продовження таблиці 1.2

Атрибут	Значення
Опис	Цей варіант використання описує механізм створення користувача існуючим адміністратором. Цей користувач зможе взаємодіяти з системою в залежності від отриманих доступів.
Актор	Адміністратор сервісу
Передумови	• Адміністратор повинен мати права адміністратора
Поточний потік	• Адміністратор складає дані для запиту, такі як: ім’я користувача, пароль, відповідні дозволи • Адміністратор відправляє запит до сервісу. • Адміністратор отримує дані створеного користувача, для подальшого використання.
Післяумови	Нового користувача створено

Таблиця 1.3 – Варіант використання UC-D2

Атрибут	Значення
Ідентифікатор	UC-D2
Назва	Видалення користувача
Опис	Цей варіант використання описує механізм видалення користувача існуючим адміністратором.
Актор	Адміністратор сервісу
Передумови	• Адміністратор повинен мати права адміністратора • Користувач, якого видаляють, повинен існувати у системі
Поточний потік	• Адміністратор складає дані для запиту, такі як: ім’я користувача. • Адміністратор відправляє запит до сервісу • Адміністратор отримує успішну відповідь про видалення відповідного користувача.
Післяумови	Користувача видалено

Таблиця 1.4 – Варіант використання UC-D3

Атрибут	Значення
Ідентифікатор	UC-D3
Назва	Корегування дозволів у користувача
Опис	Цей варіант використання описує механізм корегування дозволів користувача існуючим адміністратором. Після зміни доступів, відповідний користувач отримає дозвіл на використання нових ресурсів і функцій системи, або більше не зможе ними користуватися, в залежності від скорегованих параметрів.
Актор	Адміністратор сервісу
Передумови	• Адміністратор повинен мати права адміністратора • Користувач повинен існувати в системі.
Поточний потік	• Адміністратор складає дані для запиту, такі як: ім’я користувача, дозволи, які потрібно забрати та дозволи, які потрібно додати. • Адміністратор відправляє запит до сервісу • Адміністратор отримує успішну відповідь про зміну дозволів у користувача
Післяумови	У відповідного користувача інші дозволи.

Таблиця 1.5 – Варіант використання UC-D4

Атрибут	Значення
Ідентифікатор	UC-D4
Назва	Створення депозитного гаманця
Опис	Цей варіант використання описує механізм створення депозитного гаманця. Після створення гаманець буде автоматично очищуватись від коштів, які будуть перенесені на головну адресу.

Продовження таблиці 1.5

Атрибут	Значення
Актор	Користувач сервісу
Передумови	• Користувач повинен мати права створення гаманця
Поточний потік	• Користувач відправляє запит до сервісу • Користувач отримує успішну відповідь з даними про створений гаманець
Післяумови	Депозитний гаманець створений

Таблиця 1.6 – Варіант використання UC-D5

Атрибут	Значення
Ідентифікатор	UC-D5
Назва	Неактивний депозитний гаманець
Опис	Цей варіант використання описує механізм деактивації гаманця. Після деактивації гаманець не буде автоматично очищуватись та його баланс не буде оновлений в системі.
Актор	Користувач сервісу
Передумови	• Користувач повинен мати права деактевації гаманців. • Гаманець повинен існувати в системі. • Гаманець повинен бути активний. • Гаманець повинен мати індекс більше нуля.
Поточний потік	• Користувач складає дані для запиту, такі як: адреса депозитного гаманця. • Користувач відправляє запит до сервісу. • Користувач отримує успішну відповідь з даними про деактивований гаманець.
Післяумови	Депозитний гаманець деактивовано.

Таблиця 1.7 – Варіант використання UC-D6

Атрибут	Значення
Ідентифікатор	UC-D6
Назва	Активувати депозитний гаманець
Опис	Цей варіант використання описує механізм активації депозитного гаманця. Після активації гаманець буде автоматично очищений від коштів та баланс в системі буде скорегований.
Актор	Користувач сервісу
Передумови	• Користувач повинен мати права активації гаманців. • Гаманець повинен існувати в системі. • Гаманець повинен бути неактивний. • Гаманець повинен мати індекс більше нуля.
Поточний потік	• Користувач складає дані для запиту, такі як: адреса депозитного гаманця. • Користувач відправляє запит до сервісу. • Користувач отримує успішну відповідь з даними про активований гаманець.
Післяумови	Депозитний гаманець активовано.

Таблиця 1.8 – Варіант використання UC-D7

Атрибут	Значення
Ідентифікатор	UC-D7
Назва	Очистити депозитний гаманець від коштів
Опис	Цей варіант використання описує механізм ручного очищення коштів, якщо, по якійсь причині, автоматичний механізм не може цього зробити. Після відправки запиту потрібно дочекатися підтвердження транзакції.
Актор	Адміністратор сервісу

Продовження таблиці 1.8

Атрибут	Значення
Передумови	• Гаманець повинен існувати в системі. • Гаманець повинен мати індекс більше нуля. • Гаманець повинен мати кошти.
Поточний потік	• Адміністратор складає дані для запиту, такі як: адреса депозитного гаманця. • Адміністратор відправляє запит до сервісу. • Адміністратор отримує успішну відповідь з відправленою транзакцію у блокчейн. • Адміністратор очікує завершення підтвердження транзакції. • Адміністратор побачив підтверджену транзакцію у блокчейні.
Післяумови	Депозитний гаманець був очищений від коштів

Таблиця 1.9 – Варіант використання UC-D8

Атрибут	Значення
Ідентифікатор	UC-D8
Назва	Створити транзакцію
Опис	Цей варіант використання описує механізм ручне очищення коштів, якщо по якійсь причині автоматичний механізм не може цього зробити. Після відправки запиту потрібно дочекатися підтвердження транзакції.
Актор	Адміністратор сервісу
Передумови	• Гаманець повинен існувати в системі. • Гаманець повинен мати індекс більше нуля. • Гаманець повинен мати кошти.

Продовження таблиці 1.9

Атрибут	Значення
Поточний потік	• Адміністратор складає дані для запиту, такі як: адреса депозитного гаманця. • Адміністратор відправляє запит до сервісу. • Адміністратор отримує успішну відповідь з відправленою транзакцію у блокчейн. • Адміністратор очікує завершення підтвердження транзакції. • Адміністратор побачив підтверджену транзакцію у блокчейні.
Післяумови	Депозитний гаманець був очищений від коштів

Таблиця 1.10 – Варіант використання UC-D9

Атрибут	Значення
Ідентифікатор	UC-D9
Назва	Створення транзакції
Опис	Цей варіант використання описує механізм створення транзакції. Після створення користувачу потрібно дочекатися підтвердження транзакції у блокчейні.
Актор	Користувач сервісу
Передумови	• Користувач має відповідний доступ до функціоналу. • Гаманець повинен існувати в системі. • Гаманець повинен мати кошти. • Гаманаць має достатньо коштів для оплати комісії.

Продовження таблиці 1.10

Атрибут	Значення
Поточний потік	• Користувач складає дані для запиту, такі як: адреса гаманця, значення транзакції та швидкість транзакції. • Користувач відправляє запит до сервісу. • Користувач отримує успішну відповідь про успішне відправлення транзакції у блокчейн. • Користувач очікує завершення підтвердження транзакції. • Користувач побачив підтверджену транзакцію у блокчейні.
Післяумови	Транзакція була створена і успішно підтверджена.

Таблиця 1.11 – Варіант використання UC-D10

Атрибут	Значення
Ідентифікатор	UC-D10
Назва	Скасування транзакції, яка чекає добавлення у блок.
Опис	Цей варіант використання описує механізм скасування транзакції, яка ще не була добавлена у блок.
Актор	Користувач сервісу
Передумови	• Користувач має відповідний доступ до функціоналу. • Транзакція повина існувати в системі. • Транзакція не повина бути у блоці блокчейну (на розгляді). • Гаманець повинен мати кошти для оплати комісії.

Продовження таблиці 1.11

Атрибут	Значення
Поточний потік	• Користувач складає дані для запиту, такі як: унікальний ідентифікатор транзакції. • Користувач відправляє запит до сервісу. • Користувач отримує успішну відповідь про скасування транзакції.
Післяумови	Транзакція була скасована

Таблиця 1.12 – Варіант використання UC-D11

Атрибут	Значення
Ідентифікатор	UC-D11
Назва	Збільшення швидкості добавлення транзакції у блок
Опис	Цей варіант використання описує механізм скасування транзакції, яка ще не була добавлена у блок.
Актор	Користувач сервісу
Передумови	• Користувач має відповідний доступ до функціоналу. • Транзакція повина існувати в системі. • Транзакція не повина бути у блоці блокчейну (на розгляді). • Гаманець повинен мати кошти для оплати комісії.
Поточний потік	• Користувач складає дані для запиту, такі як: унікальний ідентифікатор транзакції. • Користувач відправляє запит до сервісу. • Користувач отримує успішну відповідь про збільшення швидкості обробки транзакції.
Післяумови	Обробка транзакції була пришвидшена

Таблиця 1.13 – Варіант використання UC-D12

Атрибут	Значення
Ідентифікатор	UC-D12
Назва	Отримання депозитних транзакцій
Опис	Цей варіант використання описує механізм отримання депозитних транзакцій. Депозитна транзакція – це транзакція, в якої отримувач це адреса любої депозитної адреси в системі.
Актор	Користувач сервісу
Передумови	• Користувач має відповідний доступ до функціоналу.
Поточний потік	• Користувач відправляє запит до сервісу. • Користувач отримує список депозитних траназакцій
Післяумови	Отримано список депозитних транзакцій.

Таблиця 1.14 – Варіант використання UC-D13

Атрибут	Значення
Ідентифікатор	UC-D13
Назва	Отримання відправлених транзакцій
Опис	Цей варіант використання описує механізм отримання депозитних транзакцій. Відправлена транзакція – це транзакція, в якої відправник це адреса нашого головного гаманця.
Актор	Користувач сервісу
Передумови	• Користувач має відповідний доступ до функціоналу.
Поточний потік	• Користувач відправляє запит до сервісу. • Користувач отримує список відправлених траназакцій
Післяумови	Отримано список відправлених транзакцій.

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Вибір моделі проектування

У відповідь на сучасні вимоги до програмного забезпечення індустрія розробки додатків зазнала значних змін, спрямованих на підвищення їх надійності, масштабованості та гнучкості. Сучасні веб-додатки покликані забезпечити високу продуктивність і здатність ефективно працювати зі змінними обсягами даних, які є типовими для ринкових реалій. Ці додатки часто побудовані на багаторівневих архітектурах, які інкапсулюють бізнес-логіку та забезпечують інтеграцію з різноманітними існуючими системами, що ускладнює управління розробкою та прогнозування результатів.

Для реалізації проекту зі значною складністю та обмеженими часовими рамками було вирішено використати методологію адаптивного проектування. Вибір адаптивного підходу відображає необхідність швидкої адаптації до мінливих вимог та умов ринку, а також забезпечує здатність проекту до ітерацій та інкрементального розвитку.

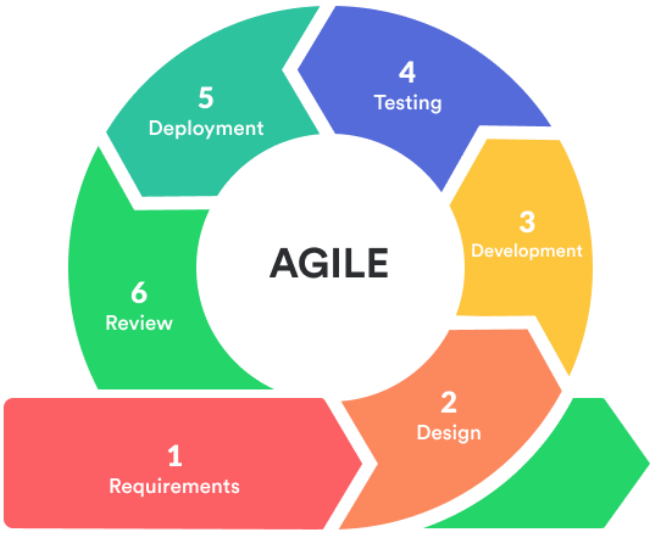


Рисунок 2.1 – Етапи процесів адаптивної методології

Були використані основні принципи адаптивної методології, серед яких ітеративний цикл розробки, постійний зворотній зв'язок із зацікавленими сторонами та пріоритет гнучкості у відповідь на зміни (див. рис. 2.1).

За основу розробки програмного комплексу було обрано гнучку методологію, яка підкреслює важливість людського фактору, комунікації, готовності продукту та здатності до змін, а не суворого дотримання планів та документації. Це дозволить проектній команді регулярно оцінювати напрямок розвитку та швидко реагувати на нові вимоги або проблеми, що виникають в процесі розробки.

Вибір гнучкої методології також обумовлений необхідністю інтеграції з блокчейном Ethereum, що вимагає глибокого розуміння технології та вміння швидко створювати прототипи. Використання гнучкої методології допомагає збалансувати потребу в швидкому розгортанні функціоналу з необхідністю постійного забезпечення якості та безпеки системи.

У цьому контексті проектування програмної системи фокусується на створенні надійної архітектури, здатної ефективно масштабуватися і адаптуватися до мінливих умов використання. Такий підхід дозволяє максимально використовувати переваги блокчейну Ethereum для вирішення конкретних бізнес-завдань, забезпечуючи при цьому високий рівень безпеки і стійкості до помилок.

2.2 Розробка бізнес моделі

Розробка бізнес-моделі веб-сервісу, який взаємодіє з блокчейном Ethereum, починається з детального аналізу функціональних вимог і ролей учасників системи. Використання діаграми акторів, важливого компонента мови UML (Unified Modeling Language), дозволяє візуалізувати взаємодію зовнішніх агентів з системою, що є критично важливим для розуміння і проектування користувацьких інтерфейсів і механізмів взаємодії.

На діаграмі акторів представлені окремі користувачі та компоненти системи, які є ключовими учасниками системи. Кожен актор асоціюється з певною роллю і відповідним набором дій, які він може виконувати в системі. Актори представлені у вигляді стилізованих фігур людей, що взаємодіють з системою, а їхні зв'язки з системою представлені асоціаціями, що описують інформаційні та керуючі потоки.

Для конкретизації функціональності системи була розроблена діаграма акторів (див. рис. 2.2), на якій показані три основні актори: Адміністратор, Клієнт (фінансова платформа) та Система. Адміністратор відповідає за управління операціями в системі та має можливість розширювати функціонал. Клієнт, який представляє фінансову платформу, виконує фінансові операції, такі як транзакції та запити на виведення коштів. Система діє як внутрішній процесор транзакцій, який реєструє їх у блокчейні Ethereum.

Рисунок 2.2 – Діаграма варіантів використання

Діаграма також ілюструє зв'язки "включати" і "розширювати", які вказують на можливість додавання або розширення функціональності в контексті більш складних операційних процесів. Наприклад, процес "Виведення коштів" може включати в себе вибір "Оптимальної ціни комісії" і може бути розширений за рахунок "Отримання даних про транзакцію". Ця діаграма є частиною проектування системи, яка допомагає визначити та узгодити функціональність системи з бізнес-процесами, а також визначити взаємодії та процеси, необхідні для забезпечення успішного та безпечного проведення фінансових транзакцій.

Розробка бізнес-моделі, яка включає в себе діаграму акторів, забезпечує стратегічне бачення і тактичне керівництво для проектної команди, спрямоване на створення веб-сервісу, який відповідає потребам користувачів і виконує бізнес-цілі в контексті динамічного середовища блокчейну.

2.2.1 Опис потоку подій

Потік подій у системі є ключовим аспектом, який визначає, як різні компоненти взаємодіють один з одним для виконання певних завдань. Розуміння цих взаємодій дозволяє не тільки оптимізувати процеси всередині системи, але й ефективно управляти потенційними помилками та забезпечити високий рівень безпеки. У цьому розділі розглядаються ключові алгоритми подій, кожен з яких представлений діаграмою послідовності, що ілюструє взаємодію між об'єктами та послідовність операцій.

Процес виведення коштів з основної адреси описаний на схемі (див. рис. 2.3) і включає наступні кроки:

- Клієнт відправляє на сервіс запит про виведення коштів;
- сервіс створює відповідні транзакції в системі та відправляє сформовану транзакцію у блокчейн;
- сервіс повертає відправлену транзакцію клієнту;

- система починає процес підтвердження транзакції за допомогою циклу перебору наступних блоків. Кількість підтверджень зростає у відповідності з просканованими блоками;
- коли кількість потрібних підтверджень була успішно зібрана, система відправляє клієнту колбек про успішне підтвердження транзакції.

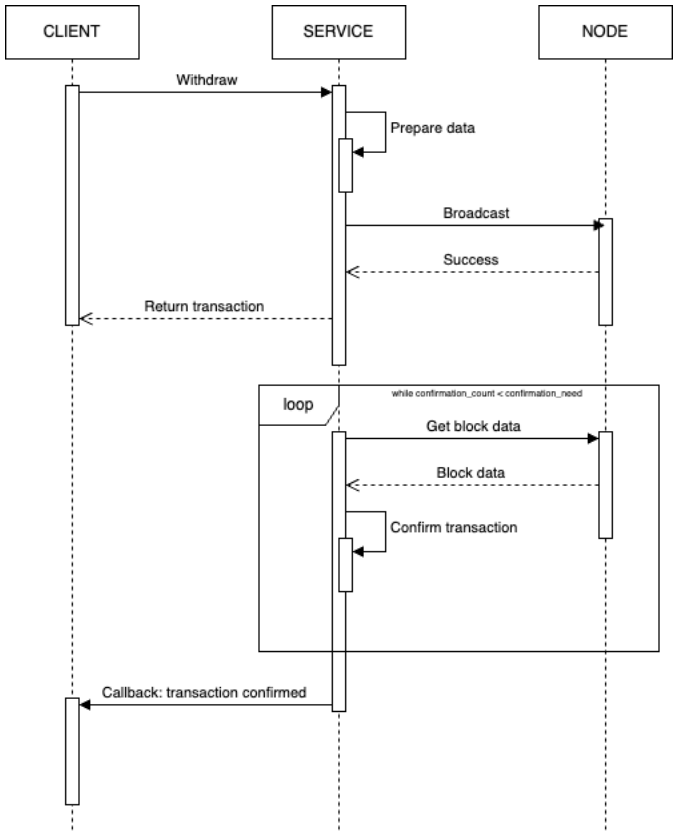


Рисунок 2.3 – Діаграма послідовності виведення коштів

Пошук депозитних транзакцій, що надходять на депозитні адреси, описано на діаграмі (див. рис. 2.4) і включає наступні кроки:

- Сервіс отримує блок з блокчейну;
- якщо адреса відправки є адресою одного з наших депозитних гаманців – робимо запис транзакції у систему;
- система починає процес підтвердження транзакції за допомогою циклу перебору наступних блоків. Кількість підтверджень зростає у відповідності з просканованими блоками;

- коли кількість потрібних підтверджень була успішно зібрана, система відправляє клієнту колбек про нову депозитну транзакцію.

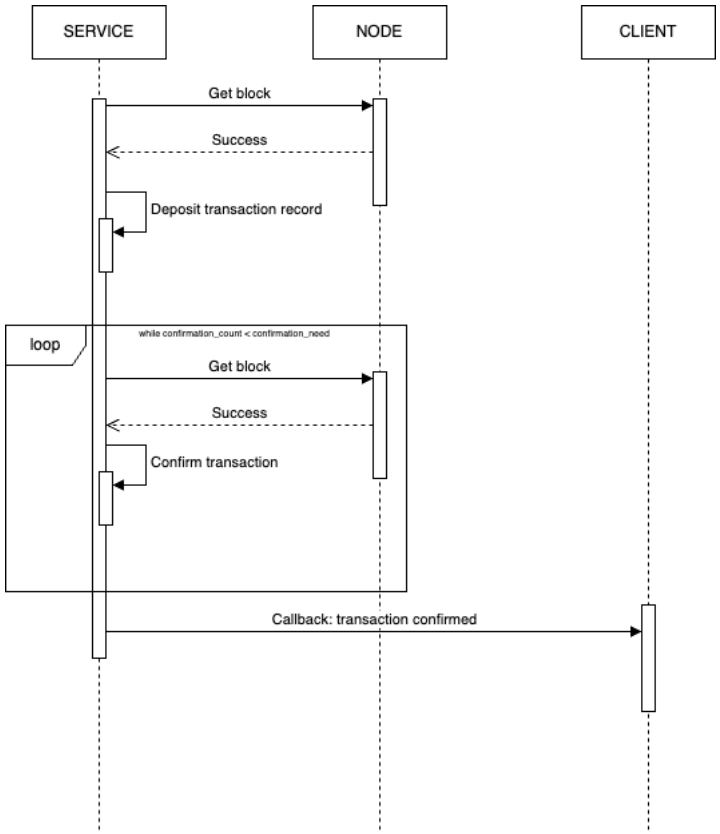


Рисунок 2.4 – Діаграма послідовності для депозитної транзакції

Механізм прискорення та відміни транзакцій [7], коли транзакція застрягла в мережі через низьку комісію, дуже схожий, тому алгоритм послідовності ідентичний і описаний на схемах (див. рис. 2.5 та див.рис. 2.5). Включає наступні кроки:

- Клієнт відправляє на сервіс запит про виведення коштів;
- сервіс створює нову транзакцію з тим самим nonce і з вищою ціною комісії;
- сервіс повертає відправлену транзакцію клієнту;
- система починає процес підтвердження транзакції за допомогою циклу перебору наступних блоків. Кількість підтверджень зростає у відповідності з просканованими блоками;

- коли кількість потрібних підтверджень була успішно зібрана, система відправляє клієнту колбек про успішне підтвердження транзакції.

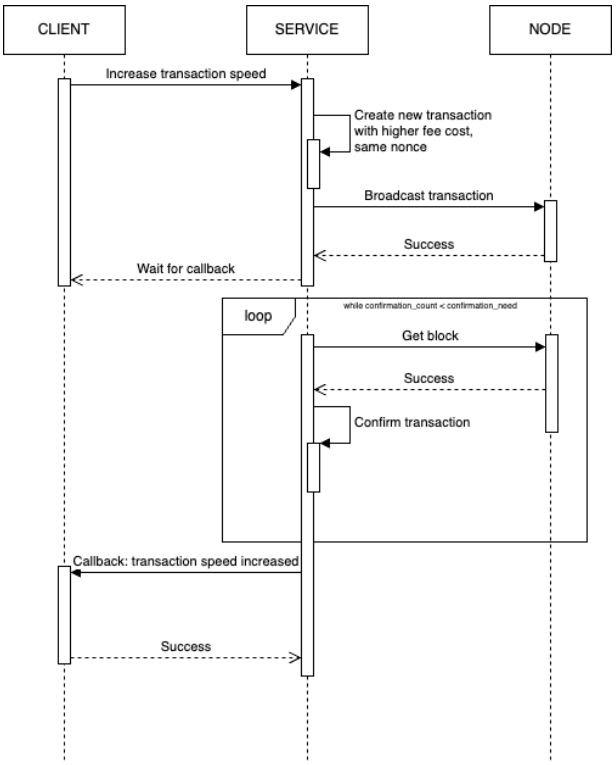


Рисунок 2.5 – Діаграма послідовності для пришвидшення транзакції

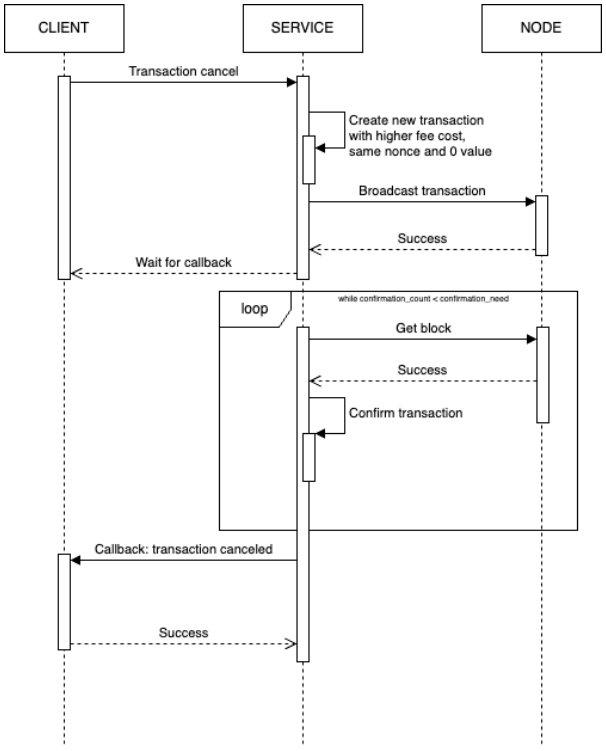


Рисунок 2.6 – Діаграма послідовності для відміни транзакції

Кліринг коштів між учасниками системи описаний на схемі (див. рис. 2.7) і включає наступні етапи:

- Система отримує депозитну транзакцію, яку записав сканер блокчейну та створює новий запис клірингу.
- Перевірка чи потрібно відправити на депозитну адресу монети для виводу коштів. Якщо потрібно – система створює транзакцію рефілу та підтверджує її.
- Система створює транзакцію трансферу, яка переведе кошти з депозитної адреси на головну та підтверджує її.
- Сервіс міняє статус клірингу на успішний.

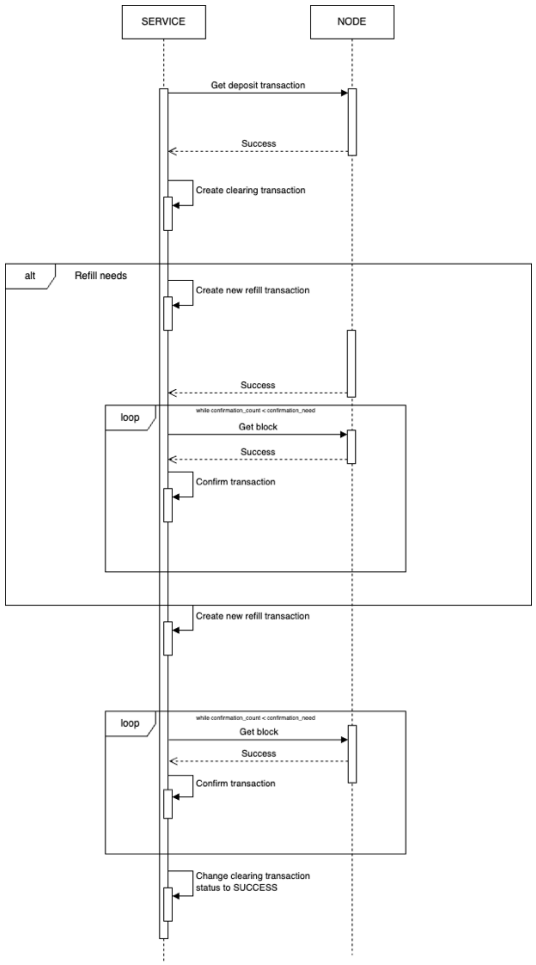


Рисунок 2.7 – Діаграма послідовності процесу клірингу

2.3 Опис мови програмування та середовища розробки

Для розробки цього проекту була обрана мова програмування Python [3] через його здатність швидко створювати прототипи, велику стандартну бібліотеку, яка надає інструменти для різних завдань, і велику кількість зовнішніх бібліотек, які можуть бути легко інтегровані в проект.

Python – це високорівнева, інтерпретована, динамічна типізована мова програмування, яку створив Гвідо ван Россум у 1991 році. Її філософія наголошує на читабельності коду та ефективності розробки, що досягається завдяки короткому та лаконічному синтаксису, який дозволяє розробникам виражати концепції у меншій кількості рядків коду, ніж це було б можливо у таких мовах, як C++ або Java. Він також підтримує багатоплатформну сумісність, що дозволяє розгортати розроблені системи на різних операційних системах. Переваги Python:

- Легко вивчати та читати: Python часто називають "виконуваним псевдокодом" через його легкий для розуміння синтаксис.
- Широкий спектр застосування: від веб-розробки до наукових досліджень та штучного інтелекту, Python надає потужні бібліотеки та фреймворки.
- Велика спільнота: Python має одну з найактивніших спільнот, яка надає значну підтримку новачкам та розробникам.
- Інтеграція: легко інтегрується з іншими мовами та інструментами, забезпечуючи високу гнучкість.

Попри всі переваги цієї мови програмування у неї є також ряд недоліків, які можуть призвести до неефективного використання часу розробника. До недоліків можна віднести наступні твердження:

- Швидкість: як інтерпретована мова, Python часто працює повільніше, ніж скомпільовані мови, хоча це можна частково вирішити за допомогою різних оптимізацій.

- Мобільна розробка: Python не є найкращим вибором для мобільної розробки порівняно з мовами, що спеціалізуються на ній.
- Споживання пам'яті: може вимагати більше пам'яті для обробки, що може бути недоліком для обмежених систем.

Важливість вибору потужного середовища розробки важко переоцінити, оскільки воно є основою для всіх етапів створення програмного продукту. Хороша IDE, така як PyCharm, надає розробникам комплексний інструментарій, який не тільки підвищує продуктивність розробки, але й гарантує високу якість кінцевого продукту. Вона сприяє кращому розумінню коду, допомагає виявляти та усувати помилки на ранніх стадіях, а також спрощує процес інтеграції з іншими інструментами та сервісами.

PyCharm, зокрема, включає такі функції, як інтелектуальне завершення коду, глибокий аналіз коду, швидке виправлення помилок, а також підтримку веб-розробки та наукових додатків. Це робить його ідеальним вибором для великих проектів і складних розробок, де потреба в гнучкості та масштабованості є критично важливою.

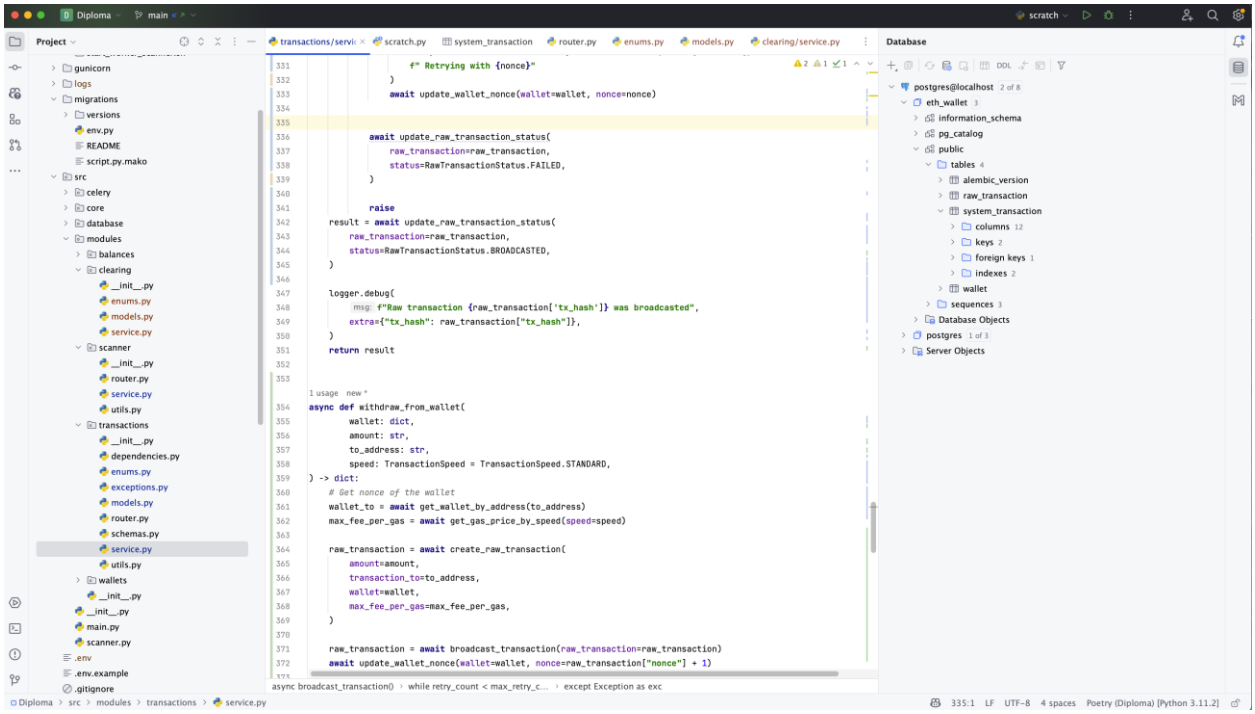


Рисунок 2.7 – Інтерфейс програми PyCharm

На рисунку 2.7 показано інтерфейс та інструментарій PyCharm, які значно полегшують життя розробникам, дозволяючи їм зосередитися на вирішенні конкретних завдань, а не на рутинному кодуванні чи конфігуруванні середовища.

Інтегрований підхід до розробки, який включає як мову програмування, так і середовище розробки, може допомогти досягти оптимального балансу між продуктивністю, масштабованістю та якістю програмного продукту.

2.4 Архітектура програмного забезпечення

Архітектура програмного забезпечення, обрана для цього проекту, орієнтована на забезпечення масштабованості, модульності та легкої інтеграції з різними системами та сервісами. Основна увага приділяється створенню чіткої та гнучкої структури проекту, що дозволяє ефективно управляти зростаючою складністю системи та реагувати на мінливі бізнес-вимоги.

Для розробки веб-сервісу для обробки REST API запитів можна виділити два важливих підходи: монолітна та мікросервісна архітектури. Кожен з них має свої особливості та варіанти використання.

2.4.1 Монолітна архітектура сервісу обробки запитів

Монолітний додаток – це однокомпонентна програма, в якій всі компоненти програми – інтерфейс користувача, бізнес-логіка, управління базами даних тощо – є частинами єдиного програмного продукту, що розгортається як єдине ціле. – є частинами єдиного програмного продукту, який розгортається як єдине ціле. Це традиційний підхід до розробки програмного забезпечення. Переваги монолітної архітектури:

- Простота розгортання: Розгортання такого додатку зазвичай вимагає менше кроків, оскільки все зводиться до одного виконуваного файлу або пакета.
- Простота розробки: Розробники можуть легко зрозуміти потік роботи програми, оскільки вона знаходиться в одному сховищі коду.
- Простота управління транзакціями: Транзакцію даних легше виконувати, оскільки всі операції з даними відбуваються в одній базі даних.

Попри всі переваги, ця архітектура має також і перелік недоліків, які можна віднести до наступних пунктів:

- Складність масштабування: Масштабування може бути складним завданням, особливо коли додаток зростає.
- Обмежена гнучкість: Внесення змін в один компонент може вплинути на весь додаток, що збільшує ризик при випуску нових версій.
- Складність відновлення після збоїв: Оскільки все знаходиться в одному блоці, помилка в одній частині може вивести з ладу весь додаток.

У цій роботі було вирішено використовувати монолітну архітектуру з модульним підходом (див. рис. 2.8). Таке рішення поєднує в собі переваги класичної монолітної архітектури з гнучкістю, яку надає модульна структура. Такий підхід забезпечує стабільність і надійність монолітного додатку, одночасно підтримуючи можливість переміщення окремих модулів в незалежні мікросервіси в міру зростання і розвитку системи [9].

Така гібридна модель дозволяє розпочати розробку як єдиного, неподільного додатку, що спрощує розгортання, тестування та управління. Кожен модуль всередині моноліту функціонує як незалежний компонент, який виконує певний набір функцій і має власний інтерфейс для взаємодії з іншими частинами системи. Це означає, що розробники можуть зосередитися на реалізації бізнес-логіки в межах одного модуля, не турбуючись про залежності та взаємодію з іншими модулями.

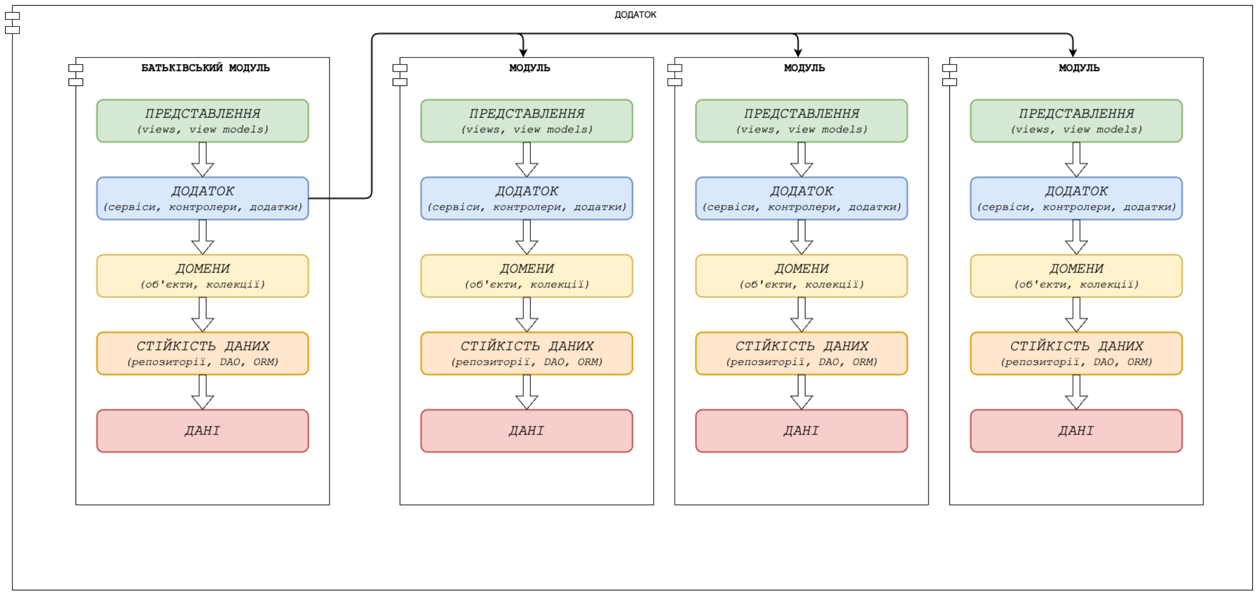


Рисунок 2.8 – Ілюстрація модульного підходу монолітної архітектури

При такому підході, коли виникає потреба в масштабуванні або вдосконаленні певних аспектів системи, окремі модулі можуть бути вилучені з моноліту і перетворені на незалежні мікросервіси. Така гнучкість забезпечує плавний перехід від монолітної до мікросервісної архітектури, дозволяючи системі розвиватися і масштабуватися без радикальної перебудови існуючого рішення [10].

Використання модульної монолітної архітектури також покращує процес розробки. Розробники можуть працювати над різними модулями паралельно без ризику виникнення конфліктів і несумісності в кодовій базі. Такий підхід підвищує продуктивність команди та скорочує час, необхідний для реалізації нових функцій та ітерацій.

2.4.2 Структура модулю

Структура модуля демонструє організований підхід до розробки програмного забезпечення, який дозволяє чітко розділити різні аспекти системи. Більш детальний опис кожного компонента модуля:

- models.py: Цей файл містить визначення моделей даних, які використовуються для взаємодії з базою даних за допомогою ORM (Object-Relational Mapping). Це дозволяє реалізувати бізнес-сутності у структурованому та зрозумілому форматі.
- schemas.py: Файл схем використовується для визначення структур даних, які надходять від користувачів або надсилаються їм. Він забезпечує валідацію та серіалізацію запитів та відповідей API, за допомогою бібліотеки Pydantic, забезпечуючи коректність обміну даними.
- enums.py: Цей файл визначає перерахування та константи, що використовуються в модулі. Вони допомагають забезпечити чистоту коду, замінюючи повторювані літеральні рядки або числа на зрозумілі іменовані константи.
- exceptions.py: Опис користувацьких винятків у файлі exceptions.py дозволяє стандартизувати обробку помилок у модулі. Це дає можливість відстежувати та коректно реагувати на специфічні ситуації, які можуть виникнути під час виконання програми.
- router.py: Маршрутизатор модуля містить визначення маршрутів, кожен з яких пов'язаний з функцією обробки запитів. Це ключовий компонент для визначення кінцевих точок API, які обробляють взаємодію з користувачем.
- service.py: Сервісний рівень містить бізнес-логіку модуля, відокремлюючи її від безпосередньої взаємодії з базою даних або користувацьким інтерфейсом. Це полегшує управління логікою обробки даних і робить код більш організованим.
- utils.py: Утиліти модуля можуть містити додаткові інструменти та допоміжні функції, які використовуються для виконання допоміжних операцій, таких як форматування даних, маніпуляції з файлами, мережеві запити тощо.

Ця модульна структура відображає чистий і масштабований підхід до розробки, де кожен компонент має чітко визначену роль і відповідальність. Такий підхід полегшує розробку програмного забезпечення, яке легше підтримувати, розширювати та адаптувати до мінливих потреб користувачів і бізнесу.

2.5 Опис функціональності модулів

Розробка будь-якого масштабного програмного забезпечення зазвичай вимагає декомпозиції загального функціоналу на менші, прості для розуміння та управління компоненти. Саме такий підхід використано в цій роботі, де кожен модуль відповідає за окремий аспект функціональності системи. Це не тільки сприяє ефективному розподілу робочого процесу між розробниками, але й забезпечує високу згуртованість та вільну взаємодію компонентів системи. Для декомпозиції системи вона була поділена на модулі, кожен з яких має своє призначення та функціонал.

Модуль «core» є основою всієї системи і містить базові сервіси, такі як конфігурація та ведення журналів. Він відповідає за ініціалізацію системи, управління ресурсами та загальне налаштування. Основні функції включають:

- Ініціалізація параметрів конфігурації
- Керування журналами роботи системи
- Надання спільних утиліт і бібліотек для інших модулів

Модуль «scanner» призначений для взаємодії з блокчейном Ethereum і забезпечує сканування мережі для підтвердження транзакцій. Він включає в себе такі функції, як:

- confirm_block – функція, котра сканує відповідний блок. Включає в себе функціонал добавлення нової транзакції у систему та підтвердження існуючих.

- confirm_pending_raw_transactions – функція, котра інкрементує число підтверджень у транзакціях, котрі знаходяться в очікувані, на основі даних конкретного блоку.
- confirm_broadcasted_transaction – функція, котра підтверджує, відправленою системою, транзакцію. Переводить транзакцію у статус очікування.
- confirm_pending_transaction – функція, котра підтверджує транзакцію в очікувані та інкрементовує кількість підтверджень.
- set_last_block – функція, котра встановлює значення останнього просканованого блоку.
- get_last_block – функція, котра дістає значення останнього просканованого блоку.

Модуль «transaction» відповідає за всі аспекти обробки транзакцій. Він надає функціонал для створення та відправлення транзакцій, обробки вхідних і вихідних платежів, керування комісіями за транзакції та перелічує наступні фунції:

- get_gas_price_by_speed – функція, котра приймає аргумент швидкості і повертає відповідне число комісії, яка буде витрачено на одну транзакцію
- get_system_transaction_by_id – функція, котра відповідає за отримання системної транзакції по унікальному ідентифікатору.
- get_raw_transaction_by_hash – функція, котра відповідає за отримання сирої транзакції по хешу транзакції.
- update_raw_transaction_status – функція, котра відповідає за оновлення статусу сирої транзакції. Так як системна транзакція напряму залежить від сирої, її статус також зміниться на відповідний.
- insert_blockchain_transaction – функція, котра вставляє транзакцію в базу даних на основі даних з блокчейну.
- drop_and_replace_tx – функція, котра заміняє стару сиру транзакцію за нову, у випадку, коли ми її скасували, або підняли ціну за комісію.

- create_raw_transaction – функція, котра відповідає за створення сирої транзакції на основі вхідних аргументів.
- create_system_transaction – функція, котра відповідає за створення системної транзакції на основі вхідних аргументів.
- broadcast_transaction – функція, котра відповідає за трансляцію сирої транзакції у блокчейн
- withdraw_from_wallet – функція, котра відповідає за виведення коштів з гаманця.
- increase_transaction_speed – функція, котра відповідає за підвищення ціни за комісію, тим самим пришвидшуючи процес добавлення у блок.
- cancel_transaction – функція, котра відповідає за скасування транзакції.

Модуль «wallets» призначений для управління гаманцями користувачів. Його основні функції включають в себе генерацію ключів та гаманців, відстеження балансів та включає в себе наступні функції:

- get_wallet_by_address – функція, котра відповідає за отримання гаманця по адресі.
- get_deposit_wallet_by_address – функція, котра відповідає за отримання гаманця у якого індекс вище нуля.
- get_wallet_by_external_id – функція, котра відповідає за отримання гаманця по унікальному ідентифікатору.
- get_wallet_by_index – функція, котра відповідає за отримання гаманця по індексу.
- get_account_by_index – функція, котра відповідає за отримання акаунту для підпису транзакції на базі вхідного гаманця.
- get_last_wallet_index – функція, котра повертає останній індекс гаманця hdwallet.
- create_wallet – функція, котра створює гаманець на базі вхідних параметрів.
- activate_wallet – функція, котра відповідальна за активацію гаманця.

- deactivate_wallet – функція, котра відповідальна за деактивацію гаманця.

Модуль «clearing» призначений для виведення коштів з депозитної адреси на головну для забезпечення головної ліквідності у системі. Може включати у себе такі процеси, як транзакція з головної адреси на депозитну та транзакція з депозитної на головну. Цей модуль включає у себе наступні функції:

- clear_deposit_wallet – функція, котра відповідає за ініціалізацію процесу очищення.
- refill_from_main_address – функція, котра відповідає за створення транзакції з головної на депозитну адресу.
- transfer_from_deposit_address – функція, котра відповідає за створення транзакції з депозитної на головну адресу.
- check_deposit_wallet_balance – функція, котра перевіряє чи достатньо коштів на гаманці для початку очищення.
- update_clearing_status – функція, котра оновлює статус очищення.

Кожен модуль є незалежною одиницею, яка взаємодіє з іншими частинами системи через чітко визначені інтерфейси, забезпечуючи таким чином легку інтеграцію та мінімізацію залежностей. Такий модульний підхід дозволяє розширювати та адаптувати систему з мінімальними зусиллями, а також полегшує майбутню підтримку та розробку продуктів.

2.6 Проектування бази даних

Проектування бази даних є ключовим елементом в архітектурі будь-якої інформаційної системи, особливо в проектах, які взаємодіють з блокчейн-технологіями, такими як Ethereum [. Ефективна структура бази даних забезпечує швидкий доступ до даних, їх безпеку, цілісність, простоту управління та масштабування. У цьому розділі ми розглянемо основні аспекти проектування

бази даних для веб-сервісу, включаючи схему бази даних, вибір СУБД, стратегію зберігання та обробки даних.

Структура бази даних була розроблена для забезпечення високої продуктивності, надійності та гнучкості системи. Вона включає наступні ключові елементи:

- Таблиця для управління системними транзакціями: забезпечують зберігання та обробку інформації про системні транзакції, їхні статуси та пов'язані з ними деталі.
- Таблиця для управління сирими транзакціями: забезпечують зберігання та обробку інформації про сирі транзакції, котрі містять у собі усю інформацію пов’язану з блокчейном.
- Таблиця для управління гаманцями: містять інформацію про гаманці, їх адреси та іншу системну інформацію.
- Таблиця для управління балансами: призначені для зберігання балансів в системі.
- Таблиці для клірингової логіки: призначені для обробки складних фінансових транзакцій, включаючи розрахунки та розподіл коштів.

Кожна з цих таблиць розроблена з урахуванням вимог до швидкості обробки даних, безпеки та масштабованості. Реляційні зв'язки між таблицями оптимізовані для забезпечення ефективного виконання запитів і зменшення навантаження на сервер.

Для візуального представлення структури бази даних була створена UML-діаграма, яка детально показує всі таблиці, їх поля, типи даних та зв'язки між ними. Ця діаграма (див. рис. 2.9) є ключовим інструментом для розуміння структури бази даних і спрощує процес розробки, тестування та подальшої підтримки системи.

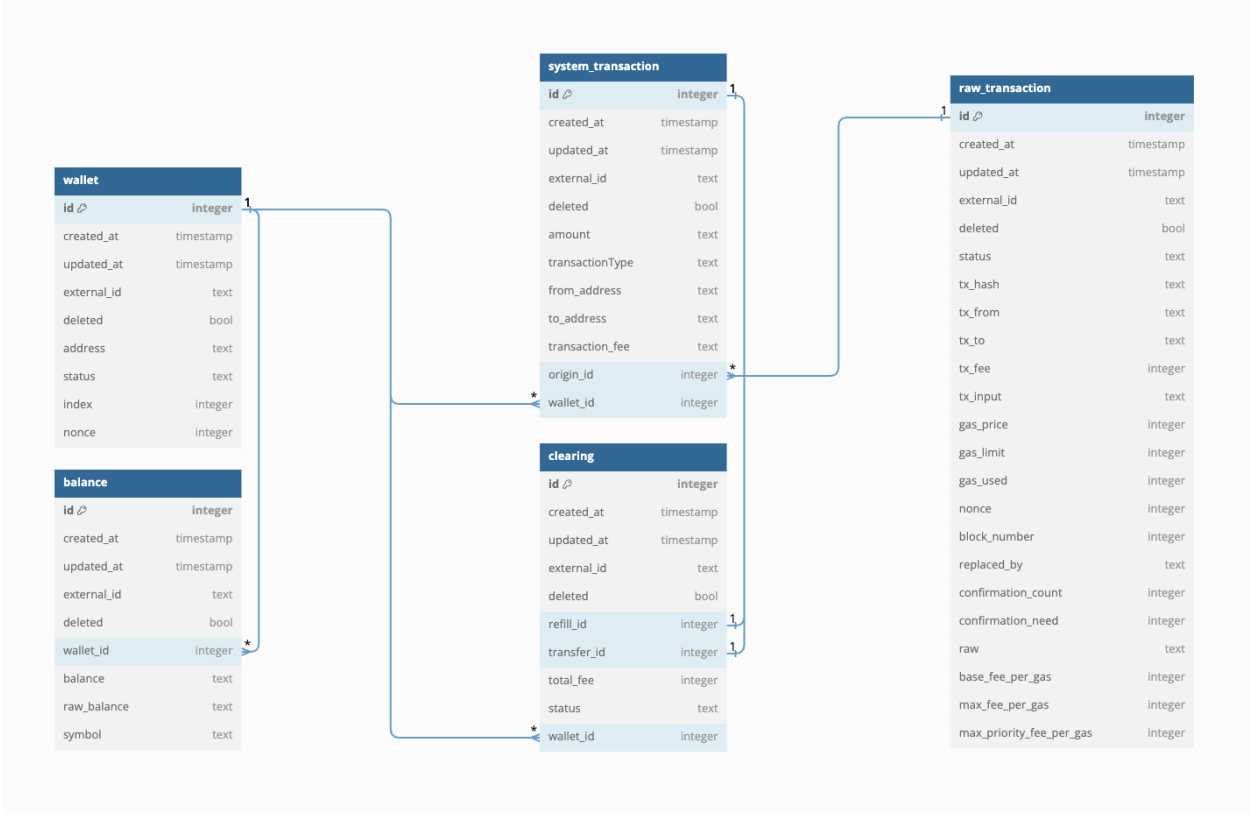


Рисунок 2.9 – Діаграма бази даних

Назви та опис наявних полів таблиці «wallet» наведено у таблиці нижче (див. табл. 2.1)

Таблиця 2.1 – Опис полів таблиці wallet

Назва	Тип	Обов’язкове поле	Опис
id	integer	Так	Унікальний ідентифікатор запису
created_at	timestamp	Так	Дата створення запису
updated_at	timestamp	Так	Дата любої зміни запису. Автоматично оновлюється при любій зміні

Продовження таблиці 2.1

Назва	Тип	Обов’язкове поле	Опис
external_id	string	Так	Унікальний ідентифікатор запису, який використовується для використання сторонніх сервісів
deleted	boolean	Так	Маркер видаленого запису
address	string	Так	Адреса гаманця
status	Enum: – ACTIVE – INACTIVE – DELETED	Так	Статус гаманця
index	integer	Так	Індекс гаманця у структурі hdwallet
nonce	integer	Так	Кількість відправлених транзакцій з гаманця

Назви та опис наявних полів таблиці «system_transaction» наведено у таблиці нижче (див. табл. 2.2).

Таблиця 2.2 – Опис полів таблиці system_transaction

Назва	Тип	Обов’язкове поле	Опис
id	integer	Так	Унікальний ідентифікатор запису
created_at	timestamp	Так	Дата створення запису

Продовження таблиці 2.2

Назва	Тип	Обов’язкове поле	Опис
updated_at	timestamp	Так	Дата любої зміни запису. Автоматично оновлюється при любій зміні
external_id	string	Так	Унікальний ідентифікатор запису
deleted	boolean	Так	Маркер видаленого запису
amount	string	Так	Значення транзакції у звичних одиницях
transaction_type	Enum: – CREATED – PENDING – CONFIRMED – FAILED	Так	Статус транзакції
from_address	string	Так	З якої адреси відбулася транзакції
to_address	string	Так	На яку адресу прийшла транзакція
transaction_fee	string	Ні	Значення комісії у звичних одиницях
origin_id	integer	Так	Унікальний ідентифікатор сирої транзакції

Назви та опис наявних полів таблиці «clearing» наведено у таблиці нижче (див. табл. 2.3).

Таблиця 2.3 – Опис полів таблиці clearing

Назва	Тип	Обов’язкове поле	Опис
id	integer	Так	Унікальний ідентифікатор запису
created_at	timestamp	Так	Дата створення запису
updated_at	timestamp	Так	Дата любої зміни запису. Автоматично оновлюється при любій зміні
external_id	string	Так	Унікальний ідентифікатор запису, який використовується для використання сторонніх сервісів
deleted	boolean	Так	Маркер видаленого запису
refill_id	integer	Ні	Унікальний ідентифікатор сирої транзакції
transfer_id	integer	Так	Унікальний ідентифікатор сирої транзакції
total_fee	integer	Ні	Значення комісії усіх транзакцій у звичних одиницях

Продовження таблиці 2.3

Назва	Тип	Обов’язкове поле	Опис
status	Enum: – CREATED – PENDING – FAILED – SUCCESS	Так	Статус очищення депозитної адреси
wallet_id	integer	Так	Унікальний ідентифікатор гаманця, який очищається

Назви та опис наявних полів таблиці «balance» наведено у таблиці нижче (див. табл. 2.4).

Таблиця 2.4 – Опис полів таблиці balance

Назва	Тип	Обов’язкове поле	Опис
id	integer	Так	Унікальний ідентифікатор запису
created_at	timestamp	Так	Дата створення запису
updated_at	timestamp	Так	Дата любої зміни запису. Автоматично оновлюється при любій зміні
external_id	string	Так	Унікальний ідентифікатор запису
deleted	boolean	Так	Маркер видаленого запису

Продовження таблиці 2.4

Назва	Тип	Обов’язкове поле	Опис
wallet_id	integer	Так	Унікальний ідентифікатор гаманця
balance	string	Так	Баланс у звичних одиницях.
raw_balance	integer	Так	Баланс у абсолютних одиницях.
symbol	string	Так	Символ балансу

Назви та опис наявних полів таблиці «raw_transaction» наведено у таблиці нижче (див. табл. 2.5).

Таблиця 2.5 – Опис полів таблиці raw_transaction

Назва	Тип	Обов’язкове поле	Опис
id	integer	Так	Унікальний ідентифікатор запису
created_at	timestamp	Так	Дата створення запису
updated_at	timestamp	Так	Дата любої зміни запису. Автоматично оновлюється при любій зміні
external_id	string	Так	Унікальний ідентифікатор запису

Продовження таблиці 2.5

Назва	Тип	Обов’язкове поле	Опис
deleted	boolean	Так	Маркер видаленого запису
status	Enum: – CREATED – BROADCAST – PENDING – CONFIRMED – FAILED	Так	
tx_hash	string	Так	Унікальний хеш транзакції
tx_from	string	Так	Адреса, яка створила транзакцію
tx_to	string	Так	Адреса куди приходить транзакція
tx_fee	integer	Ні	Значення комісії у абсолютному значенні
tx_input	string	Ні	Дані смарт-контракту
gas_price	integer	Так	Ціна газу
gas_limit	integer	Так	Ліміт газу

Продовження таблиці 2.5

Назва	Тип	Обов’язкове поле	Опис
gas_used	integer	Ні	Скільки газу було використано
nonce	integer	Так	Порядковий номер транзакції
block_number	integer	Ні	Блок, в якому знаходиться транзакція
replaced_by	string	Ні	Хеш іншої транзакції, яка замінила попередню
confirmation_count	integer	Так	Кількість потрібних підтверджень
confirmation_need	integer	Так	Кількість фактичних підтверджень
raw	string	Ні	Транзакція у хеш-представленні

2.7 Процес розвертання застосунку

Процес розгортання програми включає в себе кілька важливих кроків, щоб забезпечити встановлення, налаштування та оптимізацію програмного забезпечення для виробничого середовища. У цьому розділі детально описано послідовність дій, які необхідно виконати для успішного розгортання програми, включаючи підготовку середовища, конфігурацію інфраструктури, розподіл ресурсів і перевірку системи перед її запуском.

Для зображення процесу розгортання системи, було прийнято рішення побудувати UML-схему, зображену нижче (див. рис. 2.10).

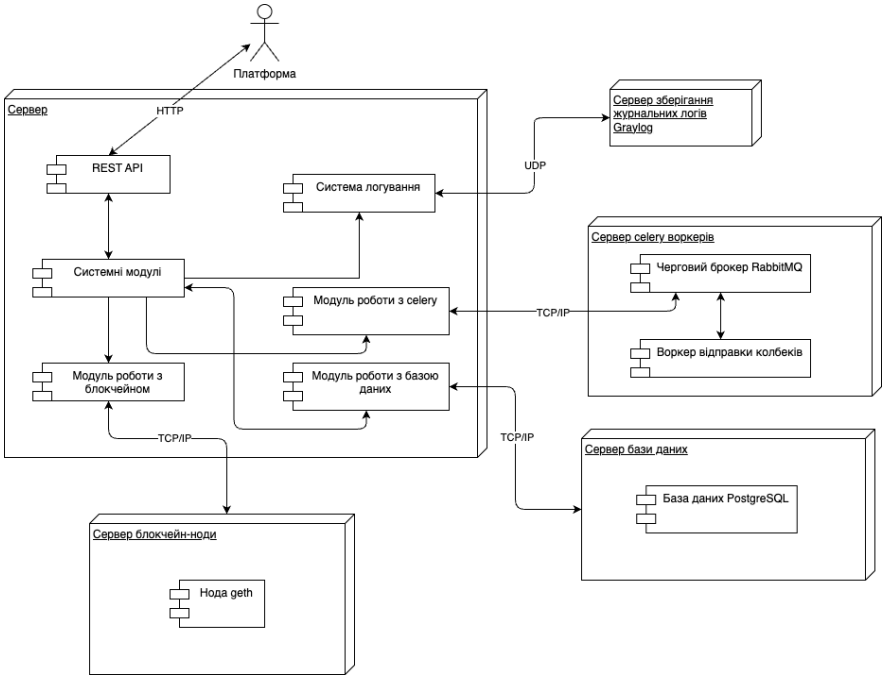


Рисунок 2.10 – Діаграма процесу розгортання

Як видно з малюнка, система, призначена для взаємодії з блокчейном Ethereum, складається з численних вузлів, які забезпечують її функціонування. Опис системи виглядає наступним чином:

- Користувач ініціює зв'язок з сервером через веб-інтерфейс, відправляючи HTTP-запит через свій веб-браузер. Сервер, у свою чергу,

обробляє цей запит через REST API, який є точкою входу для подальшої взаємодії з модулями системи.

- У системі є модуль blockchain, який підключається до вузла блокчейну через TCP/IP, використовуючи вузол Ethereum через geth для прямого зв'язку з блокчейном. Модуль celery організовує фонові завдання і процеси, в тому числі за допомогою брокера черг RabbitMQ, який забезпечує розподіл завдань між працівниками.
- Модуль бази даних взаємодіє з сервером баз даних, де в якості основної системи управління базами даних використовується PostgreSQL. Всі транзакції та пов'язані з ними дані зберігаються та управляються через цей модуль.
- Для зберігання та управління журналами системи використовується окремий сервер зберігання журналів, який використовує Graylog over UDP для агрегації та аналізу журналів.
- Крім того, система містить компонент підтримки запитів користувачів, який включає в себе модулі для обробки запитів, підтримки сесій та інших взаємодій з користувачами.

Таким чином, рисунок 2.10 демонструє взаємодію між вузлами системи та розміщеними на них компонентами, що є основою для забезпечення безперебійної роботи веб-додатку.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація ключових модулів

У цьому розділі ми зосередимося на детальному огляді реалізації ключових модулів, які лежать в основі нашої системи взаємодії з блокчейном Ethereum. Кожен модуль виконує певні функції, які разом забезпечують безперебійну та ефективну роботу всієї платформи. Значення кожного модуля базується на його здатності виконувати певні завдання, такі як управління гаманцями користувачів, обробка транзакцій, взаємодія з блокчейном та інші.

Модуль Wallets відіграє важливу роль у блокчейн-системі Ethereum. Він відповідає за управління гаманцями користувачів, генерацію ключів, захист транзакцій і підтримку безпечної взаємодії з блокчейном. Реалізація цього модуля вимагає високого рівня безпеки і точності, оскільки будь-які помилки можуть призвести до втрати коштів або інших критичних проблем.

Моделі модулю "Wallets" забезпечують представлення структур даних, необхідних для управління гаманцями. Вони включають класи для представлення гаманців та балансів (див. рис. 3.1).

```python
from sqlalchemy import BigInteger, ForeignKey, Integer, String, Text
from sqlalchemy.orm import Mapped, mapped_column

from src.database.base import BaseModel
from src.modules.wallets.enums import WalletStatus


26 usages  ± vorlov
class Wallet(BaseModel):
    __tablename__ = "wallet"

    address: Mapped[str] = mapped_column(String(255), nullable=False, unique=True)
    status: Mapped[WalletStatus] = mapped_column(
        String(50), nullable=False, default=WalletStatus.INACTIVE
    )
    nonce: Mapped[int] = mapped_column(Integer, nullable=False, default=0)
    index: Mapped[int | None] = mapped_column(Integer, nullable=True, unique=True)


8 usages  new *
class Balance(BaseModel):
    __tablename__ = "balance"

    wallet_id: Mapped[int] = mapped_column(ForeignKey("wallet.id"), nullable=False, unique=False)
    balance: Mapped[str] = mapped_column(Text, nullable=False, default="0")
    raw_balance: Mapped[int] = mapped_column(BigInteger, nullable=False, default=0)
    symbol: Mapped[str] = mapped_column(String(255), nullable=False, default="ETH")
```

Рисунок 3.1 – Моделі модулю «Wallets»

Функції модулю "Wallets" включають функціональність, необхідну для роботи з гаманцями. Це включає створення нових гаманців, генерацію, зберігання ключів та управління балансами. Сервіси також забезпечують інтеграцію з блокчейном для отримання інформації про гаманці (див. рис. 3.2).

```python
2 usages  vorlov
async def get_wallets_list(active: bool | None = None) -> list[dict]:
    query = select(Wallet)

    if active is not None:
        query = query.where(
            Wallet.status == WalletStatus.ACTIVE if active else WalletStatus.INACTIVE
        )

    return await fetch_all(query=query)


3 usages  vorlov
async def get_last_wallet_index() -> int:
    query = select(Wallet).order_by(Wallet.index.desc()).limit(1)
    wallet = await fetch_one(query=query)
    if wallet is None:
        return 0
    return wallet["index"]


3 usages  vorlov *
async def create_wallet(index: int | None = None, activate: bool = False) -> dict:
    if index is None:
        index = await get_last_wallet_index() + 1

    account = get_account_by_index(
        index=index, xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value()
    )

    # Get balance of new wallet
    raw_balance = await w3_obj.eth.get_balance(account.address)
    balance = Web3.from_wei(raw_balance, unit: "ether")

    try:
        query = (
            insert(Wallet)
            .values(
                index=index,
                address=account.address,
                status=WalletStatus.ACTIVE if activate else WalletStatus.INACTIVE,
                raw_balance=raw_balance,
                balance=balance,
            )
            .returning(Wallet)
        )

        new_wallet = await fetch_one(query=query)
```

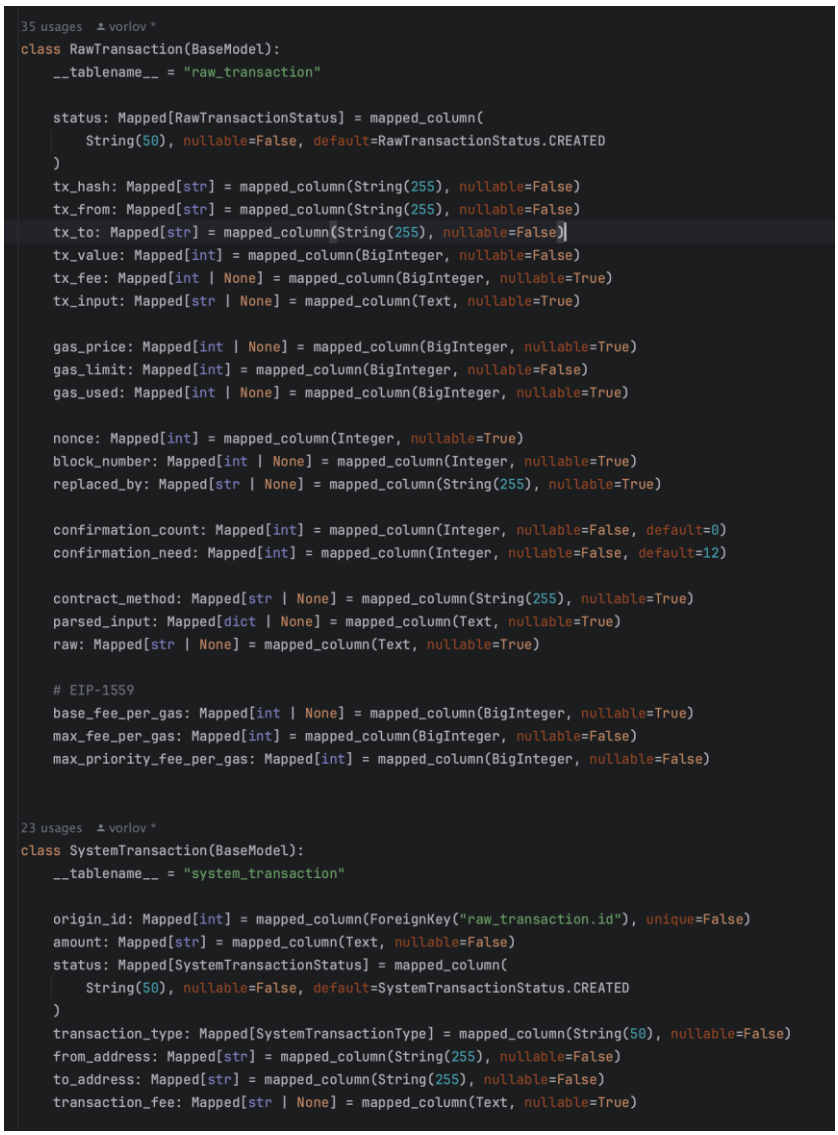
Рисунок 3.2 – Фрагмент функцій модулю «Wallets»

Для взаємодії клієнта з сервісом був розроблений інтерфейс REST API, який дозволяє виконувати стандартизовані HTTP-запити для обміну даними між користувачем та сервером. Цей інтерфейс призначений для забезпечення гнучкого та ефективного доступу до функцій системи. Для демонстрації переліку доступний шляхів було використано автозгенеровану сторінку для документації OpenAPI (див. рис. 3.3).

Рисунок 3.3 – Перелік шляхів інтерфейсу для модулю «Wallets»

Модуль "Transactions" є критичним елементом у системі взаємодії з блокчейном Ethereum, який забезпечує обробку та управління всіма транзакціями. Ця компонента системи вимагає бездоганної точності та надійності, адже помилка у цьому контексті може мати серйозні фінансові наслідки.

Моделі в модулі "Transactions" створені для репрезентації важливих даних транзакцій. Вони відображають різноманітні атрибути, які необхідні для здійснення та відстеження транзакцій, включаючи інформацію про відправника, отримувача, суму транзакції, а також статус виконання транзакції в мережі (див. рис. 3.4).



```
35 usages  vorlov *
class RawTransaction(BaseModel):
    __tablename__ = "raw_transaction"

    status: Mapped[RawTransactionStatus] = mapped_column(
        String(50), nullable=False, default=RawTransactionStatus.CREATED
    )
    tx_hash: Mapped[str] = mapped_column(String(255), nullable=False)
    tx_from: Mapped[str] = mapped_column(String(255), nullable=False)
    tx_to: Mapped[str] = mapped_column(String(255), nullable=False)
    tx_value: Mapped[int] = mapped_column(BigInteger, nullable=False)
    tx_fee: Mapped[int | None] = mapped_column(BigInteger, nullable=True)
    tx_input: Mapped[str | None] = mapped_column(Text, nullable=True)

    gas_price: Mapped[int | None] = mapped_column(BigInteger, nullable=True)
    gas_limit: Mapped[int] = mapped_column(BigInteger, nullable=False)
    gas_used: Mapped[int | None] = mapped_column(BigInteger, nullable=True)

    nonce: Mapped[int] = mapped_column(Integer, nullable=True)
    block_number: Mapped[int | None] = mapped_column(Integer, nullable=True)
    replaced_by: Mapped[str | None] = mapped_column(String(255), nullable=True)

    confirmation_count: Mapped[int] = mapped_column(Integer, nullable=False, default=0)
    confirmation_need: Mapped[int] = mapped_column(Integer, nullable=False, default=12)

    contract_method: Mapped[str | None] = mapped_column(String(255), nullable=True)
    parsed_input: Mapped[dict | None] = mapped_column(Text, nullable=True)
    raw: Mapped[str | None] = mapped_column(Text, nullable=True)

    # EIP-1559
    base_fee_per_gas: Mapped[int | None] = mapped_column(BigInteger, nullable=True)
    max_fee_per_gas: Mapped[int] = mapped_column(BigInteger, nullable=False)
    max_priority_fee_per_gas: Mapped[int] = mapped_column(BigInteger, nullable=False)


23 usages  vorlov *
class SystemTransaction(BaseModel):
    __tablename__ = "system_transaction"

    origin_id: Mapped[int] = mapped_column(ForeignKey("raw_transaction.id"), unique=False)
    amount: Mapped[str] = mapped_column(Text, nullable=False)
    status: Mapped[SystemTransactionStatus] = mapped_column(
        String(50), nullable=False, default=SystemTransactionStatus.CREATED
    )
    transaction_type: Mapped[SystemTransactionType] = mapped_column(String(50), nullable=False)
    from_address: Mapped[str] = mapped_column(String(255), nullable=False)
    to_address: Mapped[str] = mapped_column(String(255), nullable=False)
    transaction_fee: Mapped[str | None] = mapped_column(Text, nullable=True)
```

Рисунок 3.4 – Моделі модулю «Transactions»

Функції модулю "Transactions" включають в себе широкий спектр можливостей: від ініціації транзакцій до їх підпису та відправлення. Вони також надають інструменти для моніторингу статусу транзакцій та управління транзакційними зборами (див. рис. 3.5).

```python
async def drop_and_replace_transaction(
        old_raw_transaction: dict,
        new_raw_transaction: dict,
) -> dict:
    logger.info(
        msg: f"Dropping and replacing transaction {old_raw_transaction['tx_hash']}",
        extra={"tx_hash": old_raw_transaction["tx_hash"]},
    )

    if new_raw_transaction["tx_value"] == 0:
        system_transaction_status = SystemTransactionStatus.CANCELLED
    else:
        system_transaction_status = SystemTransactionStatus.PENDING

    queries = [
        update(RawTransaction)
        .where(RawTransaction.id == old_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.DROPPED_AND_REPLACED,
                "replaced_by": new_raw_transaction["tx_hash"],
            }
        ),
        update(SystemTransaction)
        .where(
            *whereclause: SystemTransaction.origin_id == old_raw_transaction["id"],
            SystemTransaction.status.notin_(SystemTransactionStatus.final_statuses()),
        )
        .values(
            {
                "status": system_transaction_status,
                "origin_id": new_raw_transaction["id"],
            }
        ),
    ]

    await execute(*queries)
    return new_raw_transaction


2 usages  vorlov *
async def insert_blockchain_transaction(blockchain_tx_data: TxData) -> None:
    # Get transaction receipt
    tx_receipt = await w3_obj.eth.get_transaction_receipt(blockchain_tx_data["hash"])
    db_raw_tx = (
        insert(RawTransaction)
```

Рисунок 3.5 – Фрагмент функцій модулю «Transactions»

REST API модулю "Transactions" відіграє ключову роль у взаємодії між клієнтом та сервером, надаючи стандартизований метод для обміну даними. Він надає набір ендпоїнтів, які дозволяють користувачам здійснювати запити на транзакції, перевіряти їх статус, та отримувати історію транзакцій. Автозгенерована документація OpenAPI [6] демонструє використання цих ендпоїнтів і слугує зручним посібником для розробників (див. рис. 3.6).

Рисунок 3.6 – REST API ендпоїнти модулю «Transactions»

Модуль "Scanner" відіграє ключову роль у моніторингу та взаємодії з блокчейном Ethereum, зокрема у скануванні блоків для виявлення та підтвердження транзакцій. Він розроблений для автоматизації процесу відстеження статусів транзакцій, що є надзвичайно важливим для забезпечення своєчасної та точної обробки платежів у системі.

Функціональність модулю "Scanner" орієнтована на виконання сканування нових блоків у блокчейні. Ключові функції включають в себе:

- Виявлення нових блоків: Неперервне сканування мережі блокчейну для виявлення нових блоків як тільки вони генеруються. Аналіз транзакцій: Відстеження та аналіз транзакцій у виявлених блоках для ідентифікації тих, що стосуються користувачів системи.
- Підтвердження транзакцій: Перевірка кількості підтверджень для кожної транзакції, щоб визначити, чи можна її вважати валідною та підтвердженою.
- Взаємодія з іншими модулями: Співпраця з модулем "Transactions" для оновлення статусу транзакцій у системі.

Модуль "Scanner" не включає в себе власних моделей або REST API ендпоїнтів, але є невід'ємною частиною бекенду, що забезпечує взаємодію з блокчейном. Він функціонує у фоновому режимі та використовує внутрішні механізми для обміну даними з іншими компонентами системи (див. рис 3.7).

```
2 usages  vorlov *
async def confirm_block(block_number: int) -> None:
    logger.info( msg: f"Confirming block {block_number}", extra={"block_number": block_number})
    block_info = await w3_obj.eth.get_block(block_number, full_transactions=True)
    logger.info(
        msg: f"Found {len(block_info.get('transactions'))} transactions in block {block_number}",
        extra={"block_number": block_number},
    )

    with signal_fence(signal.SIGINT):
        for transaction in block_info.get("transactions"):
            tx_hash = transaction.get("hash").hex()
            logger.debug(
                msg: f"Processing transaction {transaction.get('hash').hex()}",
                extra={"block_number": block_number},
            )
            transaction: TxData

            # In this stage raw transaction only have a broadcasted status

            wallet_from = await get_wallet_by_address(transaction.get("from"))
            wallet_to = await get_wallet_by_address(transaction.get("to"))

            if wallet_from or wallet_to:
                # Means that transaction is related to our system.
                # We need to create raw and system transactions.
                logger.debug(
                    msg: f"Detected transaction {tx_hash}", extra={"block_number": block_number}
                )

                broadcasted_transaction = await get_raw_transaction_by_hash(tx_hash)

                if broadcasted_transaction:
                    await confirm_raw_broadcasted_transaction(
                        raw_transaction=broadcasted_transaction,
                        blockchain_tx_data=transaction,
                    )
                else:
                    await insert_blockchain_transaction(blockchain_tx_data=transaction)

        # Confirm pending transactions
        await confirm_pending_raw_transactions(block_info)
```

Рисунок 3.7 – Фрагмент коду модулю «Scanner»

Завдяки модулю "Scanner", система може ефективно реагувати на зміни в блокчейні, а також оперативно підтверджувати транзакції, що забезпечує користувачам високий рівень довіри та надійності при роботі з блокчейн-додатками.

3.2 Тестування програмного забезпечення та оцінка якості

Після завершення розробки модулів системи, наступним кроком є детальне тестування програмного забезпечення.

Для початку ми маємо сформувати основний перелік тестових сценаріїв: TC01, TC02, TC03 та TC04. Тестові кейси відіграють вирішальну роль у розробці програмного забезпечення, оскільки вони дозволяють виявити помилки на ранній стадії. Це гарантує, що будь-які проблеми будуть вирішені до того, як програмне забезпечення потрапить до кінцевого користувача, мінімізуючи ризик надання несправного або недостатньо ефективного програмного забезпечення. Крім того, тестові кейси допомагають перевірити, що програмне забезпечення функціонує належним чином. Вони призначені для тестування конкретних функцій або частин програмного забезпечення, гарантуючи, що всі компоненти працюють правильно.

Таблиця 3.1 – Тестовий сценарій системи TC01

Ключ	Значення
Ідентифікатор	TC01
Опис	Перевірити чи правильно працює система виводу коштів з головного гаманця.
Передумови	– Головний гаманець має мати кошти

Продовження таблиці 3.1

Ключ	Значення
Кроки	– Сформувати запит на шлях /transaction/withdrawal. – Відправити запит. – Отримати успішну відповідь. – Через деякий час перевірити чи підтвердилася транзакція по шляху /transaction/{external_id}.
Результат	Результатом є успішне підтвердження транзакції у блокчейні, та зміна балансу основного гаманця. Якщо отримувач це системний гаманець – його баланс також має змінитися.

Для задовільнення тестового сценарію TC01 (див. табл. 3.1) ми формуємо та відправляємо запит через інтерфейс системи програмного забезпечення (див. рис. 3.8). Далі отримуємо успішну відповідь і очікуємо деякий час. Потім робимо запит на отримання транзакції і отримуємо успішне підтвердження (див. рис. 3.9). Для того щоб перевірити чи змінився баланс у системі ми можемо зрівняти його зі стороннім ресурсом (див. рис. 3.10)

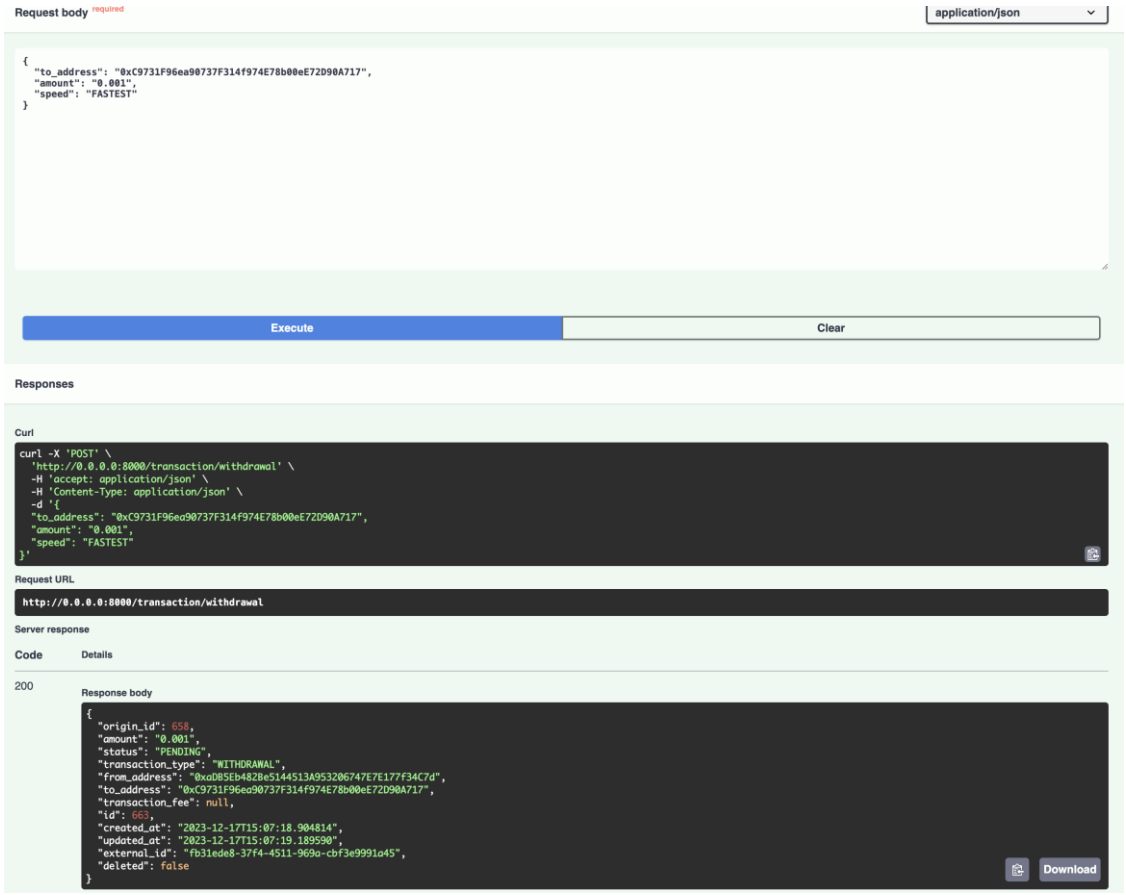


Рисунок 3.8 – Відправлення запиту для створення транзакції

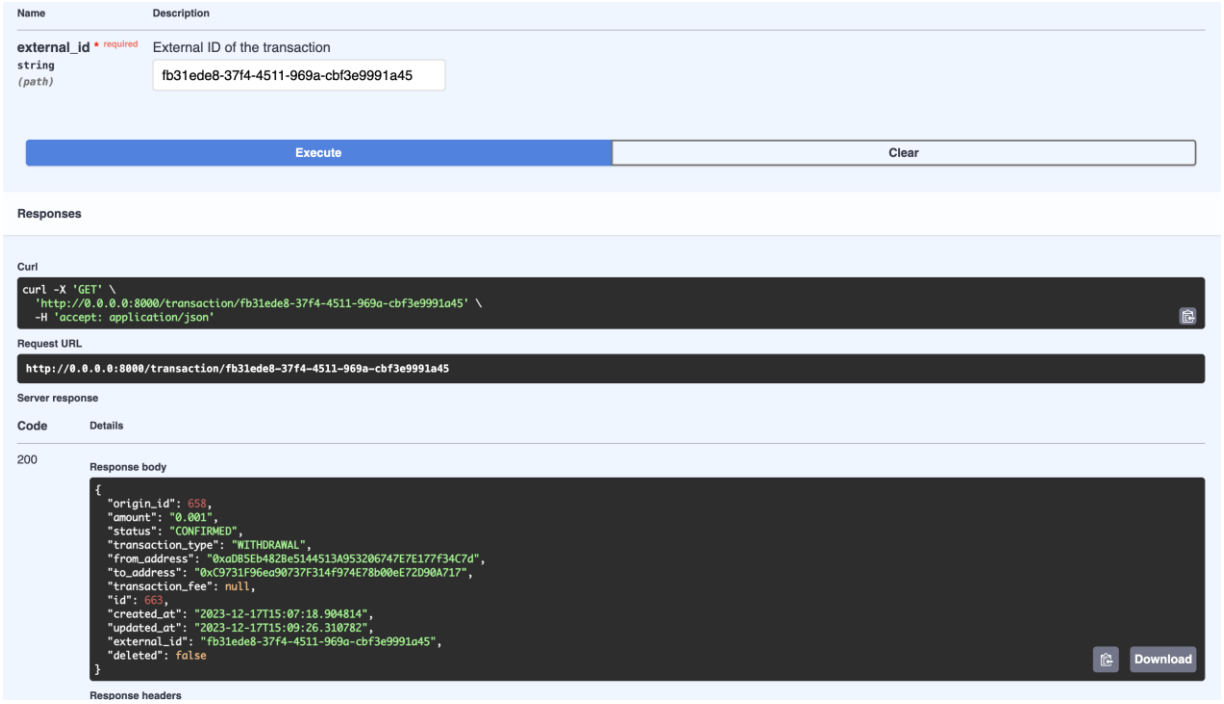


Рисунок 3.9 – Відправлення запиту на отримання тразакції

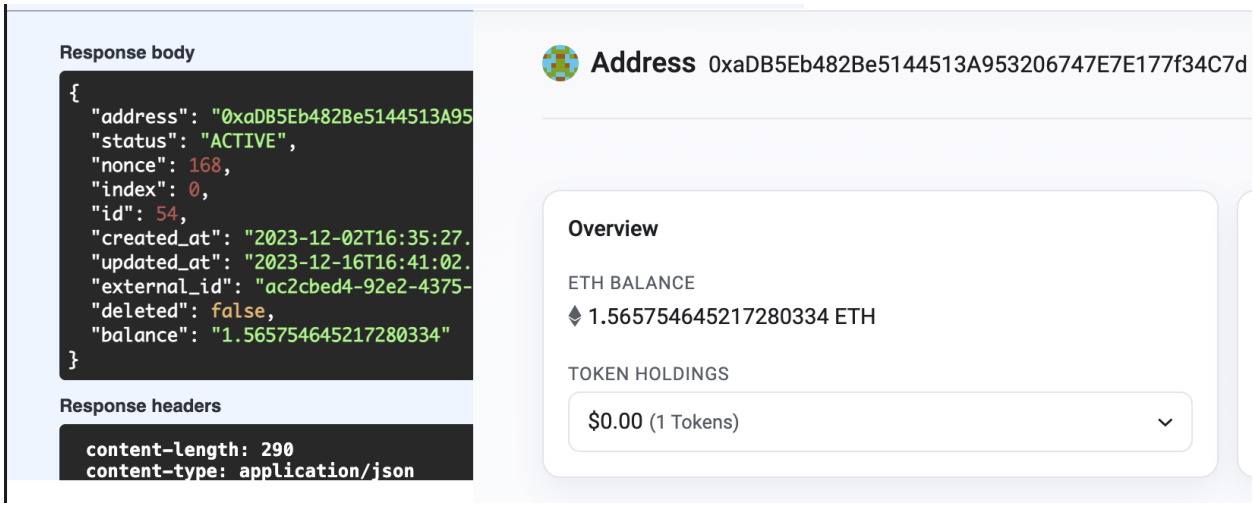


Рисунок 3.10 – Порівняння балансу у системі та на сторонньому сервісі

Таблиця 3.2 – Тестовий сценарій системи TC02

Ключ	Значення
Ідентифікатор	TC02
Опис	Перевірити чи правильно працює система отримання коштів
Передумови	– Має бути створений депозитний гаманець
Кроки	– Сформувати транзакцію на депозитний гаманець – Отримати депозитну транзакцію
Результат	Результатом є успішний трансфер коштів на головну адресу.

Для задовільнення тестового сценарію TC02 (див. табл. 3.2) потрібно ініціалізувати транзакцію на депозитну адресу. Ми можемо це зробити за допомогою функції withdrawal та вказати там адресу депозитного гаманця (див. рис. 3.11). Далі у базі даних ми маємо побачити 3 системних транзакції:

withdrawal, deposit, transfer (див. рис. 3.12). Це означає, що транзакція була розглянута як депозит та кошти були переадресовані на головну адресу.

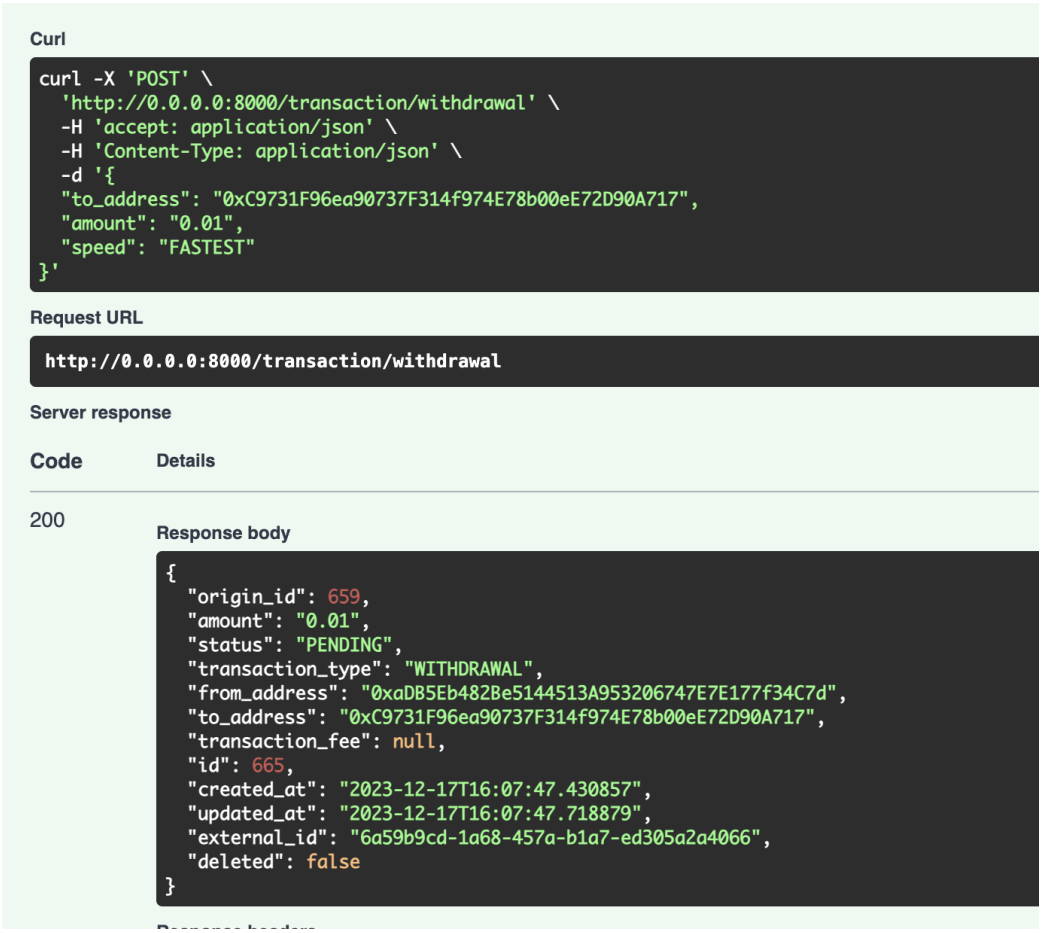


Рисунок 3.11 – Ініціалізація транзакції у системі

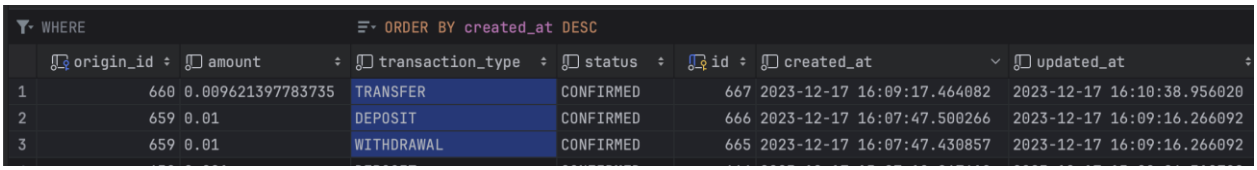


Рисунок 3.12 – Підтвердженні транзакції у базі даних

Таблиця 3.3 – Тестовий сценарій системи ТС03

Ключ	Значення
Ідентифікатор	ТС03
Опис	Перевірити чи правильно працює система скасування

Продовження таблиці 3.3

Ключ	Значення
Передумови	– Відправлена транзакція, яка ще не була додана у блокчейн.
Кроки	– Сформувати та відправити запит на скасування транзакції
Результат	Результатом є успішне скасування та змінений статус транзакції на CANCELED.

Для задовільнення тестового сценарію TC03 (див. табл. 3.3) потрібно ініціалізувати транзакцію на любу адресу (див. рис. 3.13). У нашому випадку це буде депозитний гаманець. Після цього потрібно відправити запит на скасування та перевірити чи системна транзакція була скасована у базі даних (див. рис. 3.15). Як бачимо з рисунка 3.14 у нас змінилося значення origin_id з 682 на 683 – це означає, що була створена нова транзакція скасування у базі даних.

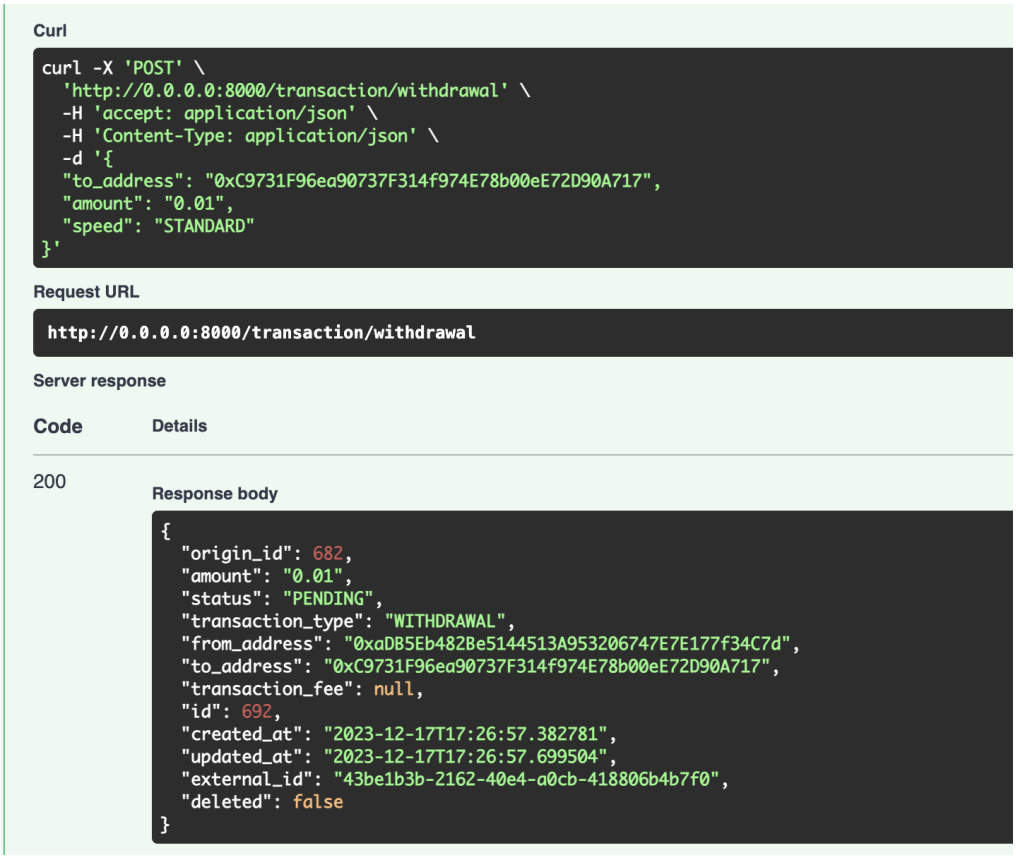


Рисунок 3.13 – Створення транзакції у системі через інтерфейс

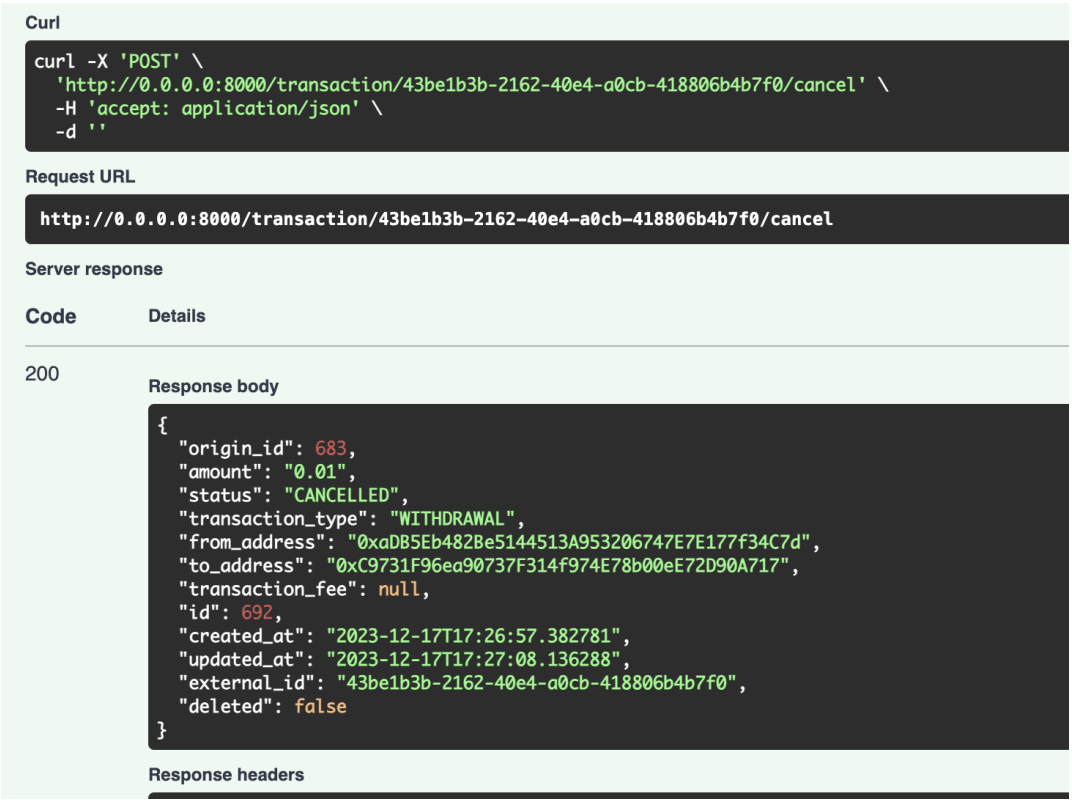
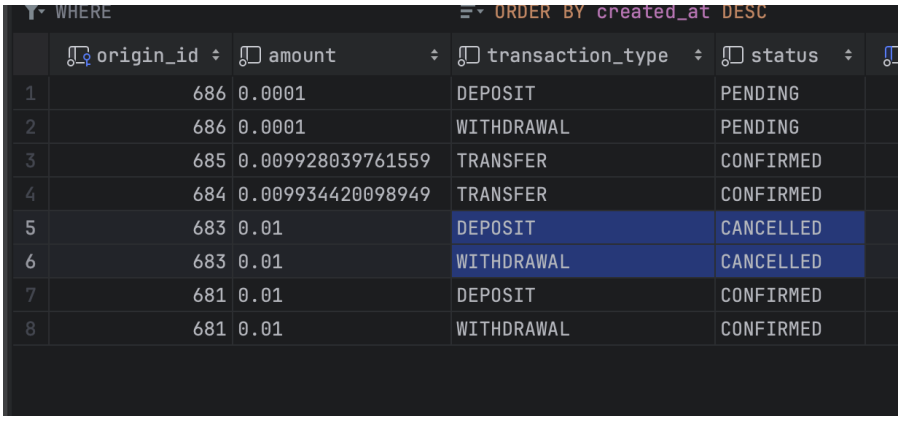


Рисунок 3.14 – Запит на відміну транзакції



WHERE ORDER BY created_at DESC

	origin_id	amount	transaction_type	status
1	686	0.0001	DEPOSIT	PENDING
2	686	0.0001	WITHDRAWAL	PENDING
3	685	0.009928039761559	TRANSFER	CONFIRMED
4	684	0.009934420098949	TRANSFER	CONFIRMED
5	683	0.01	DEPOSIT	CANCELLED
6	683	0.01	WITHDRAWAL	CANCELLED
7	681	0.01	DEPOSIT	CONFIRMED
8	681	0.01	WITHDRAWAL	CONFIRMED

Рисунок 3.15 – Стани транзакцій у базі даних

Таблиця 3.4 – Тестовий сценарій системи TC04

Ключ	Значення
Ідентифікатор	TC04
Опис	Перевірити чи правильно працює система підвищення швидкості добавлення у блок транзакції
Передумови	– Відправлена транзакція, яка ще не була додана у блокчейн.
Кроки	– Сформувати та відправити запит на підвищення швидкості транзакції
Результат	Результатом є успішне підвищення швидкості транзакції. Стара сира транзакція була замінена новою, яка містить у собі більше газу (комісії).

Для задовільнення тестового сценарію TC04 (див. табл. 3.4) ініціюємо транзакцію на адресу, що використовується як депозитний гаманець, як це показано на рисунку 3.16. Потім здійснюємо запит на скасування транзакції (див.

рис. 3.17), щоб перевірити, чи відбулося скасування в системі та відповідно у базі даних. Рисунок 3.18 демонструє, що відбулася зміна у значенні origin_id з 682 на 683, що свідчить про створення нової скасованої транзакції в базі даних.

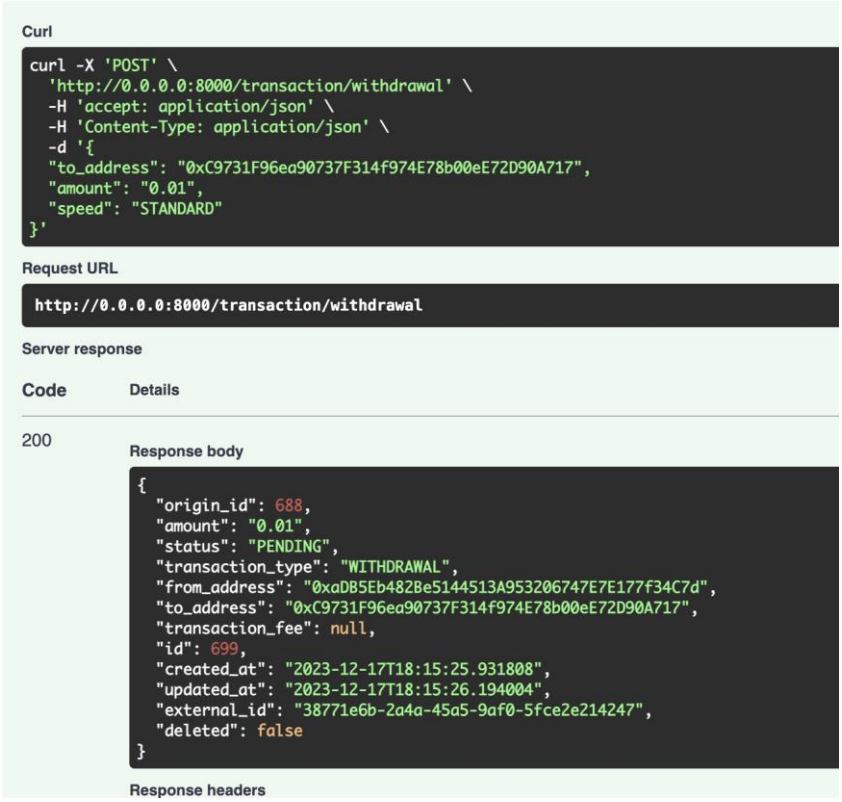


Рисунок 3.16 – Формування запиту на створення транзакції

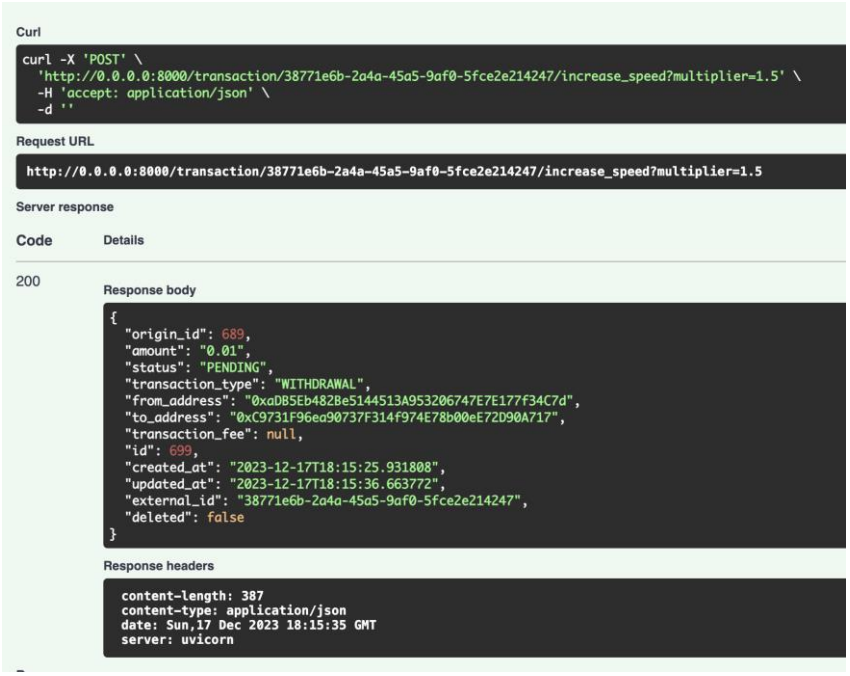


Рисунок 3.17 – Запит на пришвидшення підтвердження транзакції

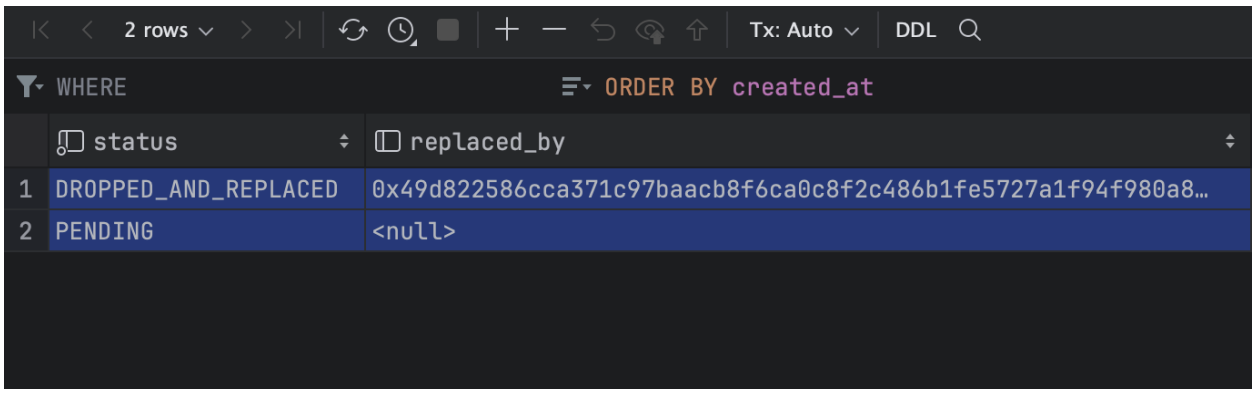


Рисунок 3.18 – Сирі транзакції у базі даних

Отже, у результаті тестування було підтверджено, що сервіс взаємодії з блокчейном Ethereum функціонує без помилок. Всі основні компоненти, включаючи модулі управління гаманцями, обробки транзакцій та сканування блоків, продемонстрували стабільну та ефективну роботу. Особливо потрібно відмітити якість реалізації модуля інтеграції з блокчейном, що забезпечує точну та своєчасну обробку транзакцій. Ефективне використання механізмів кешування сприяє високій швидкості роботи сервісу. Загалом, система успішно пройшла всі тести, демонструючи відмінну продуктивність та надійність.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Зазвичай процес розробки інтернет-магазину є довготривалим та виснажливим, так як приходиться працювати з багатьма технологіями та великим обсягом даних, через це існує ймовірність припуститися помилки. Усім добре відомо, що одним із головних методів забезпечення ефективної роботи з комп'ютером є забезпечення належних умов праці, так як при недостатньому освітлені, просторі чи неякісному дисплеї, людина відчуватиме фізичний та з часом психологічний дискомфорт, який обов'язково відобразиться на результатах роботи. Для того щоб запобігти цьому, потрібно дотримуватись ряду наказів та стандартів.

В наказі № 207 від 14.02.2018 НПАОП 0.00-7.15-18 [15], якраз описується частина вимог, яких потрібно дотримуватись при роботі з екранними пристроями.

Відповідно до третьої частини наказу [15], робоче місце працівника має відповідати наступним вимогам:

1. Робоче місце має мати достатні розміри, щоб працівник мав простір для зміни робочого положення та рухів.
2. Усе випромінювання від екранних пристроїв має бути знижене до гранично допустимого рівня з погляду безпеки та охорони здоров'я працівників.
3. Усі елементи робочого місця та їх розташування мають відповідати ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.
4. Освітлення робочого місця має створювати відповідний контраст між екраном і навколишнім середовищем та відповідати вимогам ДСанПІН 3.3.2.007-98.
5. Мікроклімат виробничих приміщень має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [16].

6. Робочий стіл чи поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, бути гнучкою під час розміщення екрана, клавіатури, документів чи устаткування.
7. Робоче крісло має бути стійким і дозволяти працівнику легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння – по висоті, та з можливістю нахилу. Для зручності слід передбачати підніжку для тих, кому це необхідно.

Також потрібно пам’ятати про безпеку, відповідно до четвертої частини наказу [15], потрібно дотримуватись наступних мінімальних вимог безпеки:

1. Перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.
2. Після закінчення роботи пристрої слід відключати від живлення.
3. При виникнення аварійної ситуації необхідно в той же час відключити пристрій від електричної мережі.
4. Не допускається:
   – Ремонтувати чи виконувати технічне обслуговування, і налагодження екранних пристроїв на робочому місці працівника під час роботи з екранними пристроями;
   – Вимикати захисні пристрої чи проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
   – Працювати з несправними екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, мигання та інші несправності.
5. Під час виконання робіт з комп’ютером, пов’язаних з нервово-емоційним напруженням мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [16].

Для забезпечення безпеки відповідно до п’ятої частини наказу «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [15], усі екранні пристрої повинні відповідати наступним мінімальним вимогам безпеки:

Відповідно до третьої частини наказу [15], робоче місце працівника розробника веб-сервісу, який працює над підтриманням стабільної роботи додатку має відповідати наступним вимогам:

1. Екранні пристрої не мають бути джерелом ризику для працівників.
2. Усе випромінювання має бути зведене до мінімального рівня з погляду безпеки і охорони здоров’я працівників.
3. Символи на дисплеї мають бути чіткими, відповідного розміру. Між символами і рядками символів повинна бути правильна відстань.
4. Зображення на дисплеї має бути стабільним, без миготінь або інших видів несправності.
5. Яскравість та контрастність символів має легко регулюватися, а також швидко адаптуватися до навколишніх умов.
6. Під час вибору монітора, слід надавати перевагу тим пристроям, які мають можливості повороту та нахилу екрану.
7. При потребі монітор може бути закріпленим на окремому столі чи підставці.
8. При виборі монітора надавайте перевагу дисплеям з матовим покриттям, щоб мінімізувати відблискування або відбивання світла.
9. При виборі клавіатури, слід надавати перевагу тій, яка відкидається і є автономною, щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук.
10. Поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання.
11. Устаткування, яке входить до робочої станції, не повинно виділяти надлишкового тепла.
12. Під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно – адаптованим до рівня знань і досвіду працівника.

Отже, для безпечної та ефективної роботи розробника веб-сервісу забезпечено належні умови праці, починаючи від робочого місця та його оснащення, та закінчуючи мікрокліматом робочого середовища, відповідно до вимог чинного законодавства.

4.2 Безпека в надзвичайних ситуаціях

Ефективність роботи людини з комп’ютером значною мірою визначається функціональним станом людини. Психофізіологічні та емоційні перенапруження, втома людини-оператора можуть призвести в комп’ютеризованих системах керування до помилок і як наслідок – до значних економічних втрат.

Згідно зі статистичними даними від 40 до 75% аварій літаків зумовлено людським фактором [17]. Відмови комп’ютеризованої системи керування рухом залізничного транспорту, на гірничо-збагачувальних комбінатах з вини операторів становлять понад 50% їх загальної кількості, причому значна їх частина спричинена невідповідністю функціонального стану оператора складності виконуваної роботи.

Трудова діяльність користувачів комп’ютерів відбувається у певному виробничому середовищі, яке впливає на їх функціональний стан. Найбільш значимі – фізичні фактори виробничого середовища, до яких належать електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ціла низка світлотехнічних показників.

Трудовий процес суттєво впливає на психофізіологічні можливості користувачів комп’ютерів, оскільки їх діяльність характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес

користувачів комп’ютерів відзначається значними інформаційними навантаженнями.

Професійні якості та виробничий досвід, які визначають внутрішні засоби діяльності, обумовлюють надійну та безпомилкову діяльність користувачів комп’ютерів, дозволяють знаходити безпечні методи розв’язання виробничих завдань навіть у нестандартних ситуаціях.

Зовнішні засоби діяльності, які в основному визначаються ергономічними показниками щодо організації робочого місця, формою та параметрами його елементів, просторового розташування основного і допоміжного устаткування, можуть суттєво знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп’ютерів.

Психологічний захист населення спрямовується на зменшення та нейтралізацію негативних психічних станів і реакцій серед населення у разі загрози та виникнення НС [19]. Основні заходи включають:

- своєчасне застосування ліцензованих та дозволених до застосування в Україні інформаційних, психопрофілактичних і психокорекційних методів впливу на особистість;
- виявлення за допомогою психологічних методів чинників, які сприяють виникненню соціально-психологічної напруженості;
- використання сучасних психологічних технологій для нейтралізації негативного впливу чинників НС на населення.

Оскільки робота користувачів комп’ютерів найчастіше проходить за активної взаємодії з іншими людьми, то виникають питання раціоналізації міжособистісних стосунків. Цей комплекс питань порушує як психологічні, так і соціально- психологічні аспекти трудових взаємовідносин, які також є факторами "ризику", що відчутно впливають на функціональний стан користувачів комп’ютерів.

Визначення та вивчення факторів, що впливають на функціональний стан користувачів комп'ютерів дозволить виділити основні причини виникнення станів напруженості, стомлення, стресу і здійснити відповідні профілактичні заходи.

Отже, до основних факторів, що впливають на функціональний стан користувачів комп’ютера належать:

1. середовище – характеризується такими шкідливими факторами:
   1.1 фізичні: електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ряд світлотехнічних показників;
   1.2 хімічні: пил, шкідливі хімічні речовини, які виділяються при роботі принтера і копіювальної техніки;
   1.3 біологічні: підвищений вміст в повітрі патогенних мікроорганізмів, особливо у приміщенні з великою кількістю працюючих, при недостатній вентиляції, особливо у період епідемій;
   1.4 психофізіологічні: напруження зору та уваги, інтелектуальні та емоційні навантаження, тривалі статичні навантаження і монотонність праці.
2. трудовий процес – характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп'ютерів відзначається значними інформаційними навантаженнями;
3. внутрішні засоби діяльності – це професійні риси та виробничий досвід, які обумовлюють надійну та безпомилкову діяльність користувачів комп'ютерів, дозволяють знаходити безпечні методи розв'язання виробничих завдань навіть у нестандартних ситуаціях;
4. зовнішні засоби діяльності – визначаються ергономічними показниками щодо організації робочого місця, форми та параметрів його елементів, просторового розташування основного і допоміжного устаткування, які можуть суттєво знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп'ютерів;
5. соціально-психологічні фактори трудових взаємовідносин.

У професійних операторів частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, серцево-судинної, імунної та статевої систем, захворювання шкіри. Зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів.

Сучасна професія користувача ВДТ належить до розумової праці, яка характеризується: високою напруженістю зорових функцій; одноманітною позою; великою кількістю стереотипних висококоординованих рухів, що виконуються лише м'язами кистей рук на фоні малої загальної рухової активності; значним нервовоемоційним компонентом, особливо в умовах дефіциту часу; роботою з великими масивами інформації, що викликає активізацію уваги та інших вищих 50 психічних функцій. Крім того, при роботі з дисплеями на електронно-променевих трубках виникає вплив на користувача цілої низки факторів фізичної природи — електростатичні поля, радіочастотне та рентгенівське випромінювання тощо.

Діяльність професіоналів можна поділити на три групи:

1. Діяльність, яка пов'язана з виконанням нескладних багаторазово повторюваних операцій, що не вимагають великого розумового напруження. Наприклад, робота операторів комп'ютерного набору, працівників довідкових служб.
2. Діяльність, яка пов'язана із здійсненням логічних операцій, що постійно повторюються. Це робота інженера-економіста, інженера-проектувальника, оператора автоматизованого виробництва.
3. Діяльність, коли в процесі роботи необхідно приймати рішення за відсутності заздалегідь відомого алгоритму. Наприклад, робота інженера програміста, диспетчерів руху залізничного транспорту, аеропортів тощо.

У користувачів, які інтенсивно використовують комп'ютер в умовах значних розумових напружень досить часто (40—70%) виникають психологічні та поведінкові порушення (нервозність, роздратування, тривога, нерішучість,

замкнутість тощо). Серед користувачів ВДТ в США і Європі значного поширення набуло специфічне захворювання, яке отримало назву синдром комп'ютерного стресу (СКС). СКС супроводжується головним болем, запаленням очей, алергією, роздратованістю, млявістю і депресією. Інформаційне перевантаження користувачів ВДТ супроводжується низкою специфічних захворювань, які називають інформаційними. Першим симптомом їх є головний біль. Дослідження, проведені в США, Німеччині, Швейцарії та інших країнах, показали, що робота з обслуговування ВДТ супроводжується підвищеним напруженням зору, інтенсивністю і монотонністю праці, збільшенням статичних навантажень, нервово-психічним напруженням, впливом різного виду випромінювань та ін. Внаслідок цього серед операторів ВДТ, як зазначають фахівці Всесвітньої організації охорони здоров'я, частіше, ніж в інших групах працюючих, трапляються такі професійні захворювання, як передчасна стомлюваність, погіршення зору, 51 м'язові і головні болі, психічні й нервові розлади, хвороби серцево-судинної системи, онкологічні захворювання та ін. Вважається, що стан організму операторів ВДТ визначається комплексним впливом факторів трудового процесу і середовища, значення яких є неоднаковим. На операторів з малим стажем роботи на ВДТ домінуючий вплив чинять фактори середовища, а на операторів зі стажем понад 5 років – фактори трудового процесу.

Комп'ютерний зоровий синдром (КЗС) – комплекс порушень здоров'я, який може виникати у користувачів персональних комп'ютерів (ПК) [18]. У користувачів ПК дуже поширені кон'юнктивіти і блефарити, патогенетично пов'язані з КЗС. Синдром розвивається при умові, що робоче місце організовано неправильно – у користувача незручне крісло, відсутні пюпітри для паперів, підставки для ніг та кистей рук, не встановлена висота і нахил монітора відносно очей, відстань від очей до екрана. За таких умов тіло людини при роботі займає вимушене положення: спина статично напружена, шия витягнута, плечі жорстко фіксовані. Напружені м'язи погіршують кровотік у сонних артеріях, а недостатнє кровозабезпечення головного мозку веде до очманіння, появи головного болю. На фоні шийного остеохондрозу з'являється відчуття випирання очних яблук, туману

в очах, мушок та райдужних кіл у полі зору. Розвитку КЗС сприяє поганий мікроклімат приміщення, значна загальна іонізація та мікробне забруднення, а також куріння.

Національною радою з наукових досліджень США для стану зорового дискомфорту був уведений термін "астенопія", який означає "будь-які суб'єктивні зорові симптоми чи емоційний дискомфорт, що є результатом зорової діяльності". Симптоми астенопії були класифіковані на "очні" (біль, печія та різь в очах, почервоніння повік та очних яблук, ломота у надбрівній частині тощо) та "зорові" (пелена перед очима, мерехтіння, швидка втома під час зорової роботи та ін.).

Таким чином, на користувача комп'ютера впливає комплекс факторів. Урахування ступеня та якості впливу цих факторів на функціональний стан дозволяють розробити заходи та засоби щодо забезпечення безпеки, підвищення працездатності та збереження здоров'я користувачів комп'ютерів.

ВИСНОВКИ

В ході виконання цієї кваліфікаційної роботи було досягнуто значних успіхів у розробці веб-сервісу для взаємодії з блокчейном Ethereum. Були досліджені та розроблені ключові модулі системи, що забезпечують управління гаманцями, обробку транзакцій та моніторинг блокчейну. В процесі розробки були створені діаграми класів та діаграми варіантів використання, що допомогли у візуалізації архітектури системи.

Проведене тестування показало високу надійність та ефективність розробленого веб-сервісу. Всі модулі системи продемонстрували стабільну роботу та точність у виконанні своїх функцій. Особлива увага була приділена безпеці та точності в реалізації модуля "Wallets", а також ефективності модуля "Scanner" у моніторингу блокчейну.

Важливо зазначити, що розробка такого роду сервісів має велике значення у сучасному світі фінансових технологій. Використання блокчейн-технологій в системі платежів і транзакцій відкриває нові можливості для безпечного, прозорого та ефективного обміну цифровими активами.

У подальшому дослідженні можна було б зосередитись на розширенні функціональності системи, зокрема, розвитку додаткових модулів для підвищення ефективності взаємодії з різними блокчейнами. Також можливе впровадження нових технологій для підвищення швидкості та надійності транзакцій.

У процесі роботи було виявлено, що інтеграція таких рішень як REST API та інші сучасні технології значно сприяє покращенню взаємодії користувача з системою, роблячи її більш гнучкою та доступною. Використання контейнеризації та мікросервісної архітектури в процесі розробки забезпечило високий рівень масштабованості та здатність до швидкого впровадження змін.

Завдяки детальному тестуванню і аналізу, було визначено потенційні напрямки для подальшого вдосконалення системи. Зокрема, велику увагу варто

приділити підвищенню безпеки транзакцій та захисту даних користувачів, а також оптимізації швидкості обробки запитів.

У цілому, розроблений веб-сервіс є вагомим внеском у розвиток блокчейн-технологій і забезпечує стабільний та надійний механізм для взаємодії з Ethereum. Перспективи розвитку даного проекту є значними, враховуючи постійно зростаючу популярність та важливість блокчейн-технологій у сучасному цифровому світі. В майбутньому робота може бути доповнена новими функціональними можливостями та вдосконалена з урахуванням новітніх технологічних трендів у сфері блокчейну, що зробить її ще більш цінною та ефективною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.
2. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник ІВ Бойко, МР Петрик, Г Цуприк – 2020
3. Python Developer Documentation [Елекронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/.
4. Bitcoin Documentation Reference [Елекронний ресурс] – Режим доступу до ресурсу: https://developer.bitcoin.org/reference/
5. Пасічник О.Г. Основи веб-дизайну [Текст]: навч.посібн. / О.Г.Пасічник, О. В. Пасічник, І. В. Стеценко. – К. : Вид. група BHV, 2009 – 336 с.
6. OpenAPI Specification Guide [Елекронний ресурс] – Режим доступу до ресурсу: https://swagger.io/specification/.
7. Ethereum Developers Documentation [Елекронний ресурс] – Режим доступу до ресурсу: https://ethereum.org/en/developers/docs/.
8. FastAPI Framework How To Guide [Елекронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/.
9. Susan J. Flower. Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization 1st Edition [Текст] / Susan J. Flower. – O’Reilly, 2016. – 124 с.
10. David Beazley, Brian K. Jones. Python Cookbook, 3rd Edition [Текст] / David Beazley. – O’Reilly, 2013. – 244 с.
11. Antony Lewis. The Basics of Bitcoins and Blockchains: An Introduction to Cryptocurrencies and the Technology that Powers Them (Cryptography,

Derivatives Investments, Futures Trading, Digital Assets, NFT) [Текст] / Antony Lewis. – Mango, 2018. – 308 с.

12. Andreas Antonopoulos, Gavin Wood Ph.D. Mastering Ethereum: Building Smart Contracts and DApps 1st Edition / Gavin Wood Ph.D. – O'Reilly Media, 2018. – 269с.
13. Пасічник В. В. Веб-технології [Текст]: підруч. / В. В. Пасічник, О.В. Пасічник, Д. І. Угрин. – Львів : Магнолія 2013. – 336 с.
14. Docker Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/develop/.
15. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#Text.
16. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [Електронний ресурс]. – 1999. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text.
17. Основні причини аварій літаків [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://suspilne.media/7904-letiti-bez-strahu-vidpovidaemo- na-najposirenisi-zapitanna-pro-aviakatastrofi/.
18. Комп'ютерний зоровий синдром [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://linza.com.ua/uk/articles/blog/kompyuternyy-zritelnyy-sindrom-simptomy-i-lechenie/.
19. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / В.С. Стручок –Тернопіль: ФОП Паляниця В. А., –156 с.

ДОДАТКИ

Додаток А – Публікація у науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.41

Орлов Володимир
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА ВЕБ-СЕРВІСУ ВЗАЄМОДІЇ З БЛОКЧЕЙНОМ ETHEREUM ДЛЯ ФІНАНСОВОЇ ПЛАТФОРМИ МОВОЮ PYTHON

Orlov Volodymyr

DEVELOPMENT OF A WEB SERVICE FOR INTERACTION WITH THE ETHEREUM BLOCKCHAIN FOR A FINANCIAL PLATFORM IN PYTHON

Актуальність теми роботи полягає у нагальній потребі сучасного цифрового світу в розробці безпечних та ефективних рішень для управління цифровими активами через блокчейн-технології. Впровадження блокчейну Ethereum у фінансову сферу та бізнес-процеси відкриває нові перспективи для безпечного обміну та управління активами, а також надає надійність та прозорість фінансових операцій.

Основне призначення розробленого веб-сервісу полягає у забезпеченні інтерфейсу для взаємодії з блокчейном Ethereum, що включає управління гаманцями, обробку транзакцій та моніторинг стану блокчейну. Такий підхід дозволяє користувачам ефективно та безпечно використовувати цифрові активи, забезпечуючи надійний механізм для їх управління та перевірки.

Об'єктом дослідження та розробки у моїй роботі є веб-сервіс, спрямований на інтеграцію із блокчейном Ethereum [1]. Важливість цього дослідження полягає у створенні рішення, яке відповідає сучасним вимогам безпеки, швидкості та ефективності у сфері цифрових фінансів.

Для реалізації проекту було використано сучасні технології та інструменти програмування, зокрема, Python, FastAPI, Docker, що забезпечують гнучкість, масштабованість та надійність рішення. Такий підхід дозволяє розробникам адаптувати систему до змінюваних вимог ринку та потреб користувачів.

У моїй роботі важливу роль відіграє технічне втілення концепції блокчейну у практично застосовні рішення, що дозволяє використовувати переваги цієї технології для підвищення безпеки, надійності та ефективності фінансових операцій.

Розвиток цього веб-сервісу становить суттєвий внесок у сферу блокчейн-технологій, особливо з урахуванням активного впровадження цифрових технологій у фінансову індустрію. Інтеграція з Ethereum відкриває безмежні можливості не тільки для ефективного управління цифровими активами, але й для розвитку нових, інноваційних підходів у фінансовому секторі. Саме тому ця робота має велике значення для ринку цифрових фінансових послуг, де вимоги до безпеки та прозорості операцій зростають з кожним днем.

Література

1. Andreas Antonopoulos, Gavin Wood Ph.D. Mastering Ethereum: Building Smart Contracts and DApps 1st Edition / Gavin Wood Ph.D. – O'Reilly Media, 2018. – 269c.

2. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – "Інженерія програмного забезпечення" для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

Лістинг коду 1 – Програмний код модулю «wallets»

```
import logging

from eth_account.signers.local import LocalAccount
from sqlalchemy import insert, select, update
from sqlalchemy.exc import IntegrityError
from web3 import Web3

from src.core.config import settings, w3_obj
from src.database.redis import redis
from src.database.utils import execute, fetch_all, fetch_one
from src.modules.wallets.enums import WalletStatus
from src.modules.wallets.exceptions import WalletWithIndexAlreadyExists
from src.modules.wallets.models import Balance, Wallet
from src.modules.wallets.utils import get_account_by_index

logger = logging.getLogger("root")


async def get_wallet_by_address(address: str) -> dict | None:
    wallet = await fetch_one(query=select(Wallet).where(Wallet.address == address))

    if wallet is None:
        return None

    wallet["account"] = get_account_by_index(
        xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value(), index=wallet["index"]
    )

    balance = await get_wallet_balance(wallet)

    return {
        **wallet,
        "balance": balance[1],
    }


async def get_deposit_wallet_by_address(address: str) -> dict | None:
    wallet = await fetch_one(
        query=select(Wallet).where(Wallet.address == address, Wallet.index > 0)
    )

    if wallet is None:
        return None

    wallet["account"] = get_account_by_index(
        xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value(), index=wallet["index"]
    )

    return wallet


async def get_wallet_by_external_id(external_id: str) -> dict | None:
    wallet = await fetch_one(query=select(Wallet).where(Wallet.external_id == external_id))

    if not wallet:
        return None

    wallet["account"] = get_account_by_index(
        xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value(), index=wallet["index"]
    )

    return wallet


async def get_wallet_by_index(index: int) -> dict | None:
    wallet = await fetch_one(query=select(Wallet).where(Wallet.index == index))

    if wallet is None:
        return None

    wallet["account"] = get_account_by_index(
        xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value(), index=wallet["index"]
    )

    return wallet
```

```
async def get_account_by_wallet(wallet: dict) -> LocalAccount:
    return get_account_by_index(
        xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value(), index=wallet["index"]
    )


async def get_wallets_list(active: bool | None = None) -> list[dict]:
    query = select(Wallet)

    if active is not None:
        query = query.where(
            Wallet.status == WalletStatus.ACTIVE if active else WalletStatus.INACTIVE
        )

    return await fetch_all(query=query)


async def get_last_wallet_index() -> int:
    query = select(Wallet).order_by(Wallet.index.desc()).limit(1)
    wallet = await fetch_one(query=query)
    if wallet is None:
        return 0
    return wallet["index"]


async def create_wallet(index: int | None = None, activate: bool = False) -> dict:
    if index is None:
        index = await get_last_wallet_index() + 1

    account = get_account_by_index(
        index=index, xprivate_key=settings.ETH_WALLET_XPRIV.get_secret_value()
    )

    # Get balance of new wallet
    raw_balance = await w3_obj.eth.get_balance(account.address)
    balance = Web3.from_wei(raw_balance, "ether")

    try:
        query = (
            insert(Wallet)
            .values(
                index=index,
                address=account.address,
                status=WalletStatus.ACTIVE if activate else WalletStatus.INACTIVE,
                raw_balance=raw_balance,
                balance=balance,
            )
            .returning(Wallet)
        )

        new_wallet = await fetch_one(query=query)

        await execute(
            insert(Balance).values(
                wallet_id=new_wallet["id"],
                raw_balance=raw_balance,
                balance=balance,
            )
        )
    except IntegrityError as exc:
        raise WalletWithIndexAlreadyExists(index) from exc

    new_wallet["account"] = account
    return new_wallet


async def get_wallet_balance(
        wallet: dict,
) -> tuple[int, str]:
    query = select(Balance).where(Balance.wallet_id == wallet["id"])
    result = await fetch_one(query=query)
    return result["raw_balance"], result["balance"]


async def align_wallet_balance(
        wallet: dict,
        raw_balance: int | None = None,
) -> dict:
```

```
    if raw_balance is None:
        raw_balance = await w3_obj.eth.get_balance(wallet["address"])

    balance = "{0:f}".format(Web3.from_wei(raw_balance, "ether"))

    query = (
        update(Balance)
        .where(Balance.wallet_id == wallet["id"])
        .values(
            raw_balance=raw_balance,
            balance=balance,
        )
        .returning(Balance)
    )

    await execute(query)

    logger.debug(
        f"Wallet {wallet['address']} balance was aligned to {balance}",
        extra={"wallet": wallet["address"]},
    )

    return wallet


async def activate_wallet(disabled_wallet: dict) -> dict | None:
    if disabled_wallet["status"] == WalletStatus.ACTIVE:
        raise ValueError("Wallet is already inactive")

    query = (
        Wallet.update()
        .where(Wallet.address == disabled_wallet["address"])
        .values(status=WalletStatus.ACTIVE)
        .returning(Wallet)
    )
    return await fetch_one(query=query)


async def deactivate_wallet(active_wallet: dict) -> dict | None:
    if active_wallet["status"] == WalletStatus.INACTIVE:
        raise ValueError("Wallet is already inactive")

    if active_wallet["index"] == 0:
        raise ValueError("Main wallet cannot be deactivated")

    query = (
        Wallet.update()
        .where(Wallet.address == active_wallet["address"])
        .values(status=WalletStatus.INACTIVE)
        .returning(Wallet)
    )

    return await fetch_one(query=query)


async def get_deposit_list() -> list[dict]:
    query = select(Wallet).where(Wallet.status == WalletStatus.ACTIVE, Wallet.index > 0)
    return await fetch_all(query=query)


async def get_wallet_nonce(wallet: dict) -> int:
    cached_nonce = await redis.get(f"EWS:{wallet['external_id']}:nonce")

    if cached_nonce is not None:
        return int(cached_nonce)

    return await w3_obj.eth.get_transaction_count(wallet["address"])


async def update_wallet_nonce(wallet: dict, nonce: int) -> None:
    wallet_id = wallet["external_id"]
    await redis.set(f"EWS:{wallet_id}:nonce", nonce)
```

Лістинг коду 2 – Програмний код модулю «transactions»

```
import logging

from eth_account.datastructures import SignedTransaction
from hexbytes import HexBytes
from sqlalchemy import insert, select, update
from web3 import Web3
from web3.types import TxData

from src.core.config import settings, w3_obj
from src.database.utils import execute, fetch_all, fetch_one
from src.modules.transactions.enums import (
    RAW_SYSTEM_TX_STATUS_MAPPING,
    RawTransactionStatus,
    SystemTransactionStatus,
    SystemTransactionType,
    TransactionSpeed,
)
from src.modules.transactions.models import RawTransaction, SystemTransaction
from src.modules.transactions.utils import get_gas_from_history
from src.modules.wallets.service import (
    get_wallet_by_address,
    get_wallet_by_index,
    get_wallet_nonce,
    update_wallet_nonce, get_wallet_balance,
)

logger = logging.getLogger("root")


async def get_gas_price_by_speed(speed: TransactionSpeed) -> int:
    min_, avg, max_ = await get_gas_from_history(w3_obj)

    policies = {
        TransactionSpeed.STANDARD: min_,
        TransactionSpeed.FAST: avg,
        TransactionSpeed.FASTEST: max_,
    }

    return policies[speed]


async def get_system_transaction_by_id(obj_id: int) -> dict:
    query = select(SystemTransaction).where(SystemTransaction.id == obj_id)
    return await fetch_one(query)


async def get_system_transaction_by_external_id(external_id: str) -> dict:
    query = select(SystemTransaction).where(SystemTransaction.external_id == external_id)
    return await fetch_one(query)


async def get_raw_transaction_by_id(obj_id: int) -> dict:
    query = select(RawTransaction).where(RawTransaction.id == obj_id)
    return await fetch_one(query)


async def get_system_transactions_by_raw_tx_and_type(
        raw_transaction: dict,
        transaction_type: SystemTransactionType,
) -> dict:
    query = select(SystemTransaction).where(
        SystemTransaction.origin_id == raw_transaction["id"],
        SystemTransaction.transaction_type == transaction_type,
    )
    return await fetch_one(query)


async def get_raw_transaction_by_external_id(external_id: str) -> dict:
    query = select(RawTransaction).where(RawTransaction.external_id == external_id)
    return await fetch_one(query)


async def get_raw_transaction_by_hash(tx_hash: str) -> dict:
    query = select(RawTransaction).where(RawTransaction.tx_hash == tx_hash)
    return await fetch_one(query)
```

```
async def get_raw_transactions_by_status(status: RawTransactionStatus) -> list[dict]:
    query = select(RawTransaction).where(RawTransaction.status == status)
    return await fetch_all(query)


async def update_raw_transaction_status(
        raw_transaction: dict,
        status: RawTransactionStatus,
):
    system_transaction_status = RAW_SYSTEM_TX_STATUS_MAPPING[status]

    queries = [
        (
            update(RawTransaction)
            .where(RawTransaction.id == raw_transaction["id"])
            .values(
                {
                    "status": status,
                }
            )
            .returning(RawTransaction)
        ),
        (
            update(SystemTransaction)
            .where(
                SystemTransaction.origin_id == raw_transaction["id"],
                SystemTransaction.status.notin_(SystemTransactionStatus.final_statuses()),
            )
            .values(
                {
                    "status": system_transaction_status,
                }
            )
        ),
    ]

    await execute(*queries)

    return await get_raw_transaction_by_hash(raw_transaction["tx_hash"])


async def drop_and_replace_transaction(
        old_raw_transaction: dict,
        new_raw_transaction: dict,
) -> dict:
    logger.info(
        f"Dropping and replacing transaction {old_raw_transaction['tx_hash']}",
        extra={"tx_hash": old_raw_transaction["tx_hash"]},
    )

    if new_raw_transaction["tx_value"] == 0:
        system_transaction_status = SystemTransactionStatus.CANCELLED
    else:
        system_transaction_status = SystemTransactionStatus.PENDING

    queries = [
        update(RawTransaction)
        .where(RawTransaction.id == old_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.DROPPED_AND_REPLACED,
                "replaced_by": new_raw_transaction["tx_hash"],
            }
        ),
        update(SystemTransaction)
        .where(
            SystemTransaction.origin_id == old_raw_transaction["id"],
            SystemTransaction.status.notin_(SystemTransactionStatus.final_statuses()),
        )
        .values(
            {
                "status": system_transaction_status,
                "origin_id": new_raw_transaction["id"],
            }
        ),
    ]

    await execute(*queries)
    return new_raw_transaction
```

```
async def insert_blockchain_transaction(blockchain_tx_data: TxData) -> None:
    # Get transaction receipt
    tx_receipt = await w3_obj.eth.get_transaction_receipt(blockchain_tx_data["hash"])
    db_raw_tx = (
        insert(RawTransaction)
        .values(
            {
                "status": RawTransactionStatus.PENDING,
                "tx_hash": blockchain_tx_data["hash"].hex(),
                "tx_from": blockchain_tx_data["from"],
                "tx_to": blockchain_tx_data["to"],
                "tx_value": blockchain_tx_data["value"],
                "tx_fee": tx_receipt["gasUsed"] * tx_receipt["effectiveGasPrice"],
                "tx_input": blockchain_tx_data["input"].hex(),
                "gas_price": blockchain_tx_data["gasPrice"],
                "gas_limit": blockchain_tx_data["gas"],
                "gas_used": tx_receipt["gasUsed"],
                "nonce": blockchain_tx_data["nonce"],
                "block_number": blockchain_tx_data["blockNumber"],
                "confirmation_count": 0,
                # "base_fee_per_gas": tx_receipt["baseFeePerGas"],
                "max_fee_per_gas": blockchain_tx_data["maxFeePerGas"],
                "max_priority_fee_per_gas": blockchain_tx_data["maxPriorityFeePerGas"],
            }
        )
        .returning(RawTransaction)
    )
    raw_transaction = await fetch_one(db_raw_tx)
    logger.debug(
        f"Raw transaction {blockchain_tx_data['hash'].hex()} was inserted into DB",
        extra={"tx_hash": blockchain_tx_data["hash"].hex()},
    )

    wallet_to = await get_wallet_by_address(blockchain_tx_data["to"])

    if wallet_to and wallet_to["index"] > 0:
        await create_deposit_transaction(raw_transaction)

    return raw_transaction


async def create_raw_transaction(
        value: int,
        transaction_to: str,
        wallet: dict,
        max_fee_per_gas: int | None = None,
        gas_price: int | None = None,
        is_eip1559: bool = True,
        nonce: int | None = None
) -> dict:
    if is_eip1559 and not max_fee_per_gas:
        raise ValueError("max_fee_per_gas is required for EIP1559")

    if not is_eip1559 and not gas_price:
        raise ValueError("gas_price is required for legacy transactions")

    logger.info(
        f"Creating raw transaction for wallet {wallet['external_id']}",
        extra={"wallet": wallet["external_id"]},
    )

    wallet_account = wallet["account"]
    from_address = wallet["address"]
    max_priority_fee_per_gas = Web3.to_wei("0.05", "gwei")
    nonce = nonce or await get_wallet_nonce(wallet=wallet)

    raw_transaction = {
        "to": Web3.to_checksum_address(transaction_to),
        "from": Web3.to_checksum_address(from_address),
        "value": value,
        "gas": 21000,
        "chainId": settings.CHAIN_ID,
        "nonce": nonce,
    }

    if not is_eip1559:
        raw_transaction["gasPrice"] = gas_price
    else:
        raw_transaction["maxFeePerGas"] = max_fee_per_gas
```

```
        raw_transaction["maxPriorityFeePerGas"] = max_priority_fee_per_gas

    logger.debug(
        f"Raw transaction was created, tx={raw_transaction}",
        extra={"wallet": wallet["external_id"]},
    )

    signed_tx: SignedTransaction = wallet_account.sign_transaction(raw_transaction)
    logger.debug(
        f"Raw transaction was signed, tx={signed_tx}",
        extra={
            "wallet": wallet["external_id"],
            "tx_hash": signed_tx.hash.hex(),
        },
    )
    db_raw_tx = (
        insert(RawTransaction)
        .values(
            {
                "status": RawTransactionStatus.CREATED,
                "tx_hash": signed_tx.hash.hex(),
                "tx_from": wallet["address"],
                "tx_to": transaction_to,
                "tx_value": value,
                "gas_limit": raw_transaction["gas"],
                "gas_price": gas_price,
                "nonce": nonce,
                "max_fee_per_gas": max_fee_per_gas,
                "max_priority_fee_per_gas": max_priority_fee_per_gas,
                "raw": signed_tx.rawTransaction.hex(),
            }
        )
        .returning(RawTransaction)
    )

    logger.debug(
        "Raw transaction was inserted into DB",
        extra={"wallet": wallet["external_id"], "tx_hash": signed_tx.hash.hex()},
    )

    return await fetch_one(db_raw_tx)


async def create_transaction(
        raw_transaction: dict,
        transaction_type: SystemTransactionType,
) -> dict:
    amount = Web3.from_wei(raw_transaction["tx_value"], "ether")
    status = RAW_SYSTEM_TX_STATUS_MAPPING[raw_transaction["status"]]
    tx_fee = (
        str(Web3.from_wei(raw_transaction["tx_fee"], "ether"))
        if raw_transaction["tx_fee"]
        else None
    )

    system_transaction = (
        insert(SystemTransaction)
        .values(
            {
                "origin_id": raw_transaction["id"],
                "amount": str(amount),
                "status": status,
                "transaction_type": transaction_type,
                "from_address": raw_transaction["tx_from"],
                "to_address": raw_transaction["tx_to"],
                "transaction_fee": tx_fee,
            }
        )
        .returning(SystemTransaction)
    )
    result = await fetch_one(system_transaction)
    logger.debug(
        f"System transaction was created, tx={result}",
        extra={"tx_hash": raw_transaction["tx_hash"]},
    )

    return result


async def broadcast_transaction(
```

```
        transaction: dict,
        max_retry_count: int = 3,
) -> dict:
    retry_count = 0
    raw_transaction = await get_raw_transaction_by_id(transaction["origin_id"])

    if not raw_transaction:
        raise ValueError(f"Raw transaction {transaction['origin_id']} not found")

    while retry_count < max_retry_count:
        try:
            await w3_obj.eth.send_raw_transaction(HexBytes(raw_transaction["raw"]))
            wallet_from = await get_wallet_by_address(raw_transaction["tx_from"])
            if wallet_from:
                await update_wallet_nonce(wallet=wallet_from, nonce=raw_transaction["nonce"] + 1)

            break
        except Exception as exc:
            logger.warning(
                f"Raw transaction {raw_transaction['tx_hash']} wasn't broadcasted, reason: {exc}",
                extra={"tx_hash": raw_transaction["tx_hash"]},
            )

            wallet = await get_wallet_by_address(raw_transaction["tx_from"])

            if "nonce too low" in str(exc) or "nonce is too low" in str(exc):
                nonce = await w3_obj.eth.get_transaction_count(wallet["address"])

                logger.warning(
                    f"Nonce ({wallet['nonce']} is too low for wallet {wallet['address']}."
                    f" Retrying with {nonce}"
                )
                await update_wallet_nonce(wallet=wallet, nonce=nonce)

            elif "replacement transaction underpriced" in str(exc):
                nonce = wallet["nonce"] + 1

                logger.warning(
                    f"Replacement transaction underpriced for wallet {wallet['address']}."
                    f" Retrying with {nonce}"
                )
                await update_wallet_nonce(wallet=wallet, nonce=nonce)

            await update_raw_transaction_status(
                raw_transaction=raw_transaction,
                status=RawTransactionStatus.FAILED,
            )

            raise
    await update_raw_transaction_status(
        raw_transaction=raw_transaction,
        status=RawTransactionStatus.BROADCASTED,
    )

    logger.debug(
        f"Raw transaction {raw_transaction['tx_hash']} was broadcasted",
        extra={"tx_hash": raw_transaction["tx_hash"]},
    )
    return await get_system_transaction_by_id(transaction["id"])


async def create_withdraw_transaction(
        to_address: str,
        amount: str,
        speed: TransactionSpeed = TransactionSpeed.STANDARD,
):
    # We always withdraw from the main wallet
    wallet_from = await get_wallet_by_index(0)

    # Get nonce of the wallet
    max_fee_per_gas = await get_gas_price_by_speed(speed=speed)
    raw_transaction = await create_raw_transaction(
        value=Web3.to_wei(amount, "ether"),
        transaction_to=to_address,
        wallet=wallet_from,
        max_fee_per_gas=max_fee_per_gas,
    )

    transaction = await create_transaction(
        raw_transaction=raw_transaction,
```

```
        transaction_type=SystemTransactionType.WITHDRAWAL,
    )

    # If we make WITHDRAWAL to our wallet, we need to create DEPOSIT transaction for it
    wallet_to = await get_wallet_by_address(to_address)
    if wallet_to:
        await create_deposit_transaction(raw_transaction=raw_transaction)

    transaction = await broadcast_transaction(transaction)
    return transaction


async def create_refill_transaction(
        deposit_wallet_to: dict,
        speed: TransactionSpeed = TransactionSpeed.FASTEST,
):
    amount = await get_gas_price_by_speed(speed=TransactionSpeed.FASTEST)
    refill_amount = amount * 2  # multiply by 2 to be sure that we will have enough gas

    # We always refill from the main wallet
    wallet_from = await get_wallet_by_index(0)

    # Get nonce of the wallet
    max_fee_per_gas = await get_gas_price_by_speed(speed=speed)
    raw_transaction = await create_raw_transaction(
        value=refill_amount,
        transaction_to=deposit_wallet_to["address"],
        wallet=wallet_from,
        max_fee_per_gas=max_fee_per_gas,
    )

    transaction = await create_transaction(
        raw_transaction=raw_transaction,
        transaction_type=SystemTransactionType.REFILL,
    )

    transaction = await broadcast_transaction(transaction)
    return transaction


async def create_transfer_transaction(
        deposit_wallet_from: dict,
        amount: str | None = None
):
    main_wallet = await get_wallet_by_index(0)

    if not amount:
        value = await w3_obj.eth.get_balance(deposit_wallet_from["address"])
    else:
        value = Web3.to_wei(amount, "ether")

    # Get nonce of the wallet
    max_fee_per_gas = await get_gas_price_by_speed(speed=TransactionSpeed.FASTEST)
    raw_balance, _ = await get_wallet_balance(deposit_wallet_from)

    withdrawal_value = value - max_fee_per_gas * 21000

    if withdrawal_value < 0:
        logger.warning(
            f"Wallet {deposit_wallet_from['address']} doesn't have enough funds to pay fee
({withdrawal_value})"
        )
        return

    raw_transaction = await create_raw_transaction(
        value=withdrawal_value,
        transaction_to=main_wallet["address"],
        wallet=deposit_wallet_from,
        max_fee_per_gas=max_fee_per_gas,
    )

    transaction = await create_transaction(
        raw_transaction=raw_transaction,
        transaction_type=SystemTransactionType.TRANSFER,
    )

    transaction = await broadcast_transaction(transaction=transaction)
    return transaction
```

```
async def create_deposit_transaction(raw_transaction: dict):
    wallet_to = await get_wallet_by_address(raw_transaction["tx_to"])

    if not wallet_to:
        raise ValueError(f"Wallet {raw_transaction['tx_to']} not found")

    transaction = await create_transaction(
        raw_transaction=raw_transaction,
        transaction_type=SystemTransactionType.DEPOSIT,
    )
    return transaction


async def increase_speed_of_transaction(
        transaction: dict,
        multiplier: float = 1.5,
) -> dict:
    old_raw_transaction = await get_raw_transaction_by_id(transaction["origin_id"])

    if old_raw_transaction["status"] != RawTransactionStatus.BROADCASTED:
        raise ValueError(f"Transaction {transaction['id']} is not in broadcasted status")

    from_wallet = await get_wallet_by_address(transaction["from_address"])
    if not from_wallet:
        raise ValueError(
            f"You can't increase speed of transaction {transaction['id']} because wallet not found"
        )

    old_raw_transaction = await get_raw_transaction_by_id(transaction["origin_id"])
    wallet = await get_wallet_by_address(transaction["from_address"])

    new_raw_transaction = await create_raw_transaction(
        value=old_raw_transaction["tx_value"],
        transaction_to=old_raw_transaction["tx_to"],
        wallet=wallet,
        gas_price=int(old_raw_transaction["max_fee_per_gas"] * multiplier),
        nonce=old_raw_transaction["nonce"],
        is_eip1559=False,
    )

    try:
        await w3_obj.eth.send_raw_transaction(HexBytes(new_raw_transaction["raw"]))
    except Exception as e:
        logger.error(f"Error while cancelling transaction {transaction['id']}: {e}")
        q = (
            update(RawTransaction)
            .where(RawTransaction.id == new_raw_transaction["id"])
            .values(
                {
                    "status": RawTransactionStatus.FAILED,
                }
            )
        )
        await execute(q)
        raise

    # Update statuses after broadcast
    queries = [
        update(RawTransaction)
        .where(RawTransaction.id == old_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.DROPPED_AND_REPLACED,
                "replaced_by": new_raw_transaction["tx_hash"],
            }
        ),
        update(RawTransaction)
        .where(RawTransaction.id == new_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.BROADCASTED,
            }
        ),
        update(SystemTransaction)
        .where(
            SystemTransaction.origin_id == old_raw_transaction["id"],
            SystemTransaction.status.notin_(SystemTransactionStatus.final_statuses()),
        )
        .values(
            {
```

```
                "origin_id": new_raw_transaction["id"],
                "status": SystemTransactionStatus.PENDING,
            }
        ),
    ]
    await execute(*queries)
    return await get_system_transaction_by_id(transaction["id"])


async def cancel_transaction(transaction: dict) -> dict:
    old_raw_transaction = await get_raw_transaction_by_id(transaction["origin_id"])

    if old_raw_transaction["status"] != RawTransactionStatus.BROADCASTED:
        raise ValueError(f"Transaction {transaction['id']} is not in broadcasted status")

    from_wallet = await get_wallet_by_address(transaction["from_address"])
    if not from_wallet:
        raise ValueError(
            f"You can't cancel transaction {transaction['id']} because wallet not found"
        )

    wallet = await get_wallet_by_address(transaction["from_address"])
    new_raw_transaction = await create_raw_transaction(
        value=0,
        transaction_to=wallet["address"],  # send to self
        wallet=wallet,
        gas_price=int(old_raw_transaction["max_fee_per_gas"] * 1.5),
        nonce=old_raw_transaction["nonce"],
        is_eip1559=False,
    )

    try:
        await w3_obj.eth.send_raw_transaction(HexBytes(new_raw_transaction["raw"]))
    except Exception as e:
        logger.error(f"Error while cancelling transaction {transaction['id']}: {e}")
        q = (
            update(RawTransaction)
            .where(RawTransaction.id == new_raw_transaction["id"])
            .values(
                {
                    "status": RawTransactionStatus.FAILED,
                }
            )
        )
        await execute(q)
        raise

    queries = [
        update(RawTransaction)
        .where(RawTransaction.id == old_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.DROPPED_AND_REPLACED,
                "replaced_by": new_raw_transaction["tx_hash"],
            }
        ),
        update(RawTransaction)
        .where(RawTransaction.id == new_raw_transaction["id"])
        .values(
            {
                "status": RawTransactionStatus.BROADCASTED,
            }
        ),
        update(SystemTransaction)
        .where(
            SystemTransaction.origin_id == old_raw_transaction["id"],
            SystemTransaction.status.notin_(SystemTransactionStatus.final_statuses()),
        )
        .values(
            {
                "origin_id": new_raw_transaction["id"],
                "status": SystemTransactionStatus.CANCELLED,
            }
        ),
    ]

    await execute(*queries)
    return await get_system_transaction_by_id(transaction["id"])
```

Додаток Г – Диск із кваліфікаційною роботою магістра